Computer Shop

Ian Stewart
Robin Jones

SINCLAIR ZX SPECTRUM

Programmieren leicht gemacht



Springer Basel AG

Computer Shop
Band 3

Ian Stewart / Robin Jones

Sinclair ZX Spectrum

Programmieren leichtgemacht

Aus dem Englischen von Tony Westermayr

Springer Basel AG

Die Originalausgabe erschien 1982 unter dem Titel:
"Easy Programming for the ZX Spectrum"
bei Shiva Publishing Ltd., Nantwich, England

Professor Ian Stewart
Mathematics Institute
University of Warwick
Coventry CA4, 7AL
England, U.K.

Professor Robin Jones
Computer Unit
South Kent College of Technology
Ashford, Kent
England, U.K.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Stewart, Ian:
Sinclair ZX spectrum : programmieren leicht gemacht / Ian Stewart ; Robin Jones. Aus d. Engl. von Tony Westermayr.

(Computer-Shop ; Bd. 3)
Einheitssacht.: Easy programming for the ZX spectrum ⟨dt.⟩
ISBN 978-3-7643-1491-0 ISBN 978-3-0348-6747-4 (eBook)
DOI 10.1007/978-3-0348-6747-4

NE: Jones, Robin:; GT

Ursprünglich erschienen bei Birkhäuser Verlag, Basel 1983

Umschlaggestaltung: Konrad Bruckmann

ISBN 978-3-7643-1491-0

Inhalt

Vorwort

Clive Sinclair hat es erneut geschafft, und wir auch.

Nicht zufrieden damit, mehr als eine Million Mikrocomputer ZX81 verkauft zu haben, hat er jetzt den ZX Spectrum herausgebracht. Sie können damit nicht nur einen Heimcomputer zu einem vernünftigen Preis bekommen, sondern erhalten dazu noch Ton, Farbe und hochauflösende Grafik. (Und eine akzeptable Tastatur!)

Als der ZX81 herauskam, verfaßten wir dazu einen Einführungsband, der den Titel Sinclair ZX81, Programme, Spiele, Grafik trug. Wir wollten dieses Buch hier schon "Der Sohn von Sinclair ZX81" nennen, aber das wäre doch zu albern gewesen. In gewissem Sinn stimmt es aber. Es ist die Spectrum-Version von "Sinclair ZX81, Programme, Spiele . . ."

Der Spectrum hat mit dem ZX81 verschiedene Dinge gemeinsam; wo das möglich war, haben wir sie auf die gleiche Weise behandelt. Wenn Sie also das erste Buch erworben haben und nun zu den Aufsteigern gehören, werden Sie rund ein Viertel bis ein Drittel des Inhalts wiedererkennen. Aber sogar das haben wir neu gefaßt, um die überlegenen Eigenschaften des Spectrum nutzen zu können. Der Rest ist neu, aber auf ähnliche Weise abgefaßt.

Das Ziel ist klar: Auf einfache, leicht verständliche Weise jene Eigenschaften des ZX Spectrum zu beschreiben, die ein Benutzer (wir nennen ihn künftig "User", wie das Brauch ist) kennen muß. Zusätzlich haben wir rund 3Ø Listings von "Fertigprogrammen" angefügt, die nach Belieben eingegeben und gefahren werden können; im Text finden sich noch einmal rund 3Ø Programme. Unser Vorteil dem Handbuch gegenüber: Wir können auswählen und uns auf die weniger ausgefallenen Merkmale beschränken: man sieht sonst vor lauter Bäumen den Wald nicht mehr.

Was kriegen Sie also? Sie bekommen eine ruhige Einführung für BASIC-Programmieren, Programmierstil, Farbe, Ton, bewegte Grafik, hochauflösende (raffinierte) Grafik, Debugging (bei Laien heißt das "Fehlersuche"), Zahlenverarbeitung und Umgang mit Strings (zu deutsch: "Ketten"). Wenn Sie das gelesen haben, wird das Handbuch von Sinclair für Sie eine Kleinigkeit sein. Alles ist in leichtverdauliche Abschnitte aufgeteilt, damit Sie zwei Stunden am Spectrum verbringen und anschließend das Gefühl haben können, wirklich etwas geleistet zu haben. *Und* dazu eine Vielzahl von Programmen zu Ihrer Belustigung; sie nutzen die bemerkenswerten Fähigkeiten des Spectrum wirklich aus. Beachten Sie auch die Fehlermeldungen auf Seite 192. Schneiden Sie diese aus dem Buch aus und kleben Sie sie auf ein Stück Karton. Das beiliegende Datenblatt enthält die wichtigsten Tabellen des Handbuchs. Halten Sie diese beiden Hilfsmittel stets in Reichweite. Sie ersparen sich damit viel zeitraubendes Herumblättern.

Bis jetzt haben wir uns kollektiv als "wir" bezeichnet, aber später funktioniert das nicht mehr so gut. Persönliche Erfahrungen, die durch ein "wir" vermittelt werden, wirken sonderbar. Da jeden Abschnitt nur jeweils einer von uns geschrieben hat, einigten wir uns auf das Wort "ich", mit dem wir beide gemeint sind. Falls Sie das stören sollte, bedenken Sie, wie oft sich einzelne Verfasser als "wir" bezeichnen.

1 BASIS-BASIC

ROM, RAM, REM . . . Computersprache ist gar nicht rätselhaft – das sieht nur so aus.

Computer haben im Grunde nichts Rätselhaftes an sich. Sie sind schlicht Maschinen, die Folgen von Befehlen ausführen. Die Art und *Weise* freilich, wie sie das tun, ist in der Tat geheimnisvoll, zumindest für den, der nicht gerade ein Doktorat in Festkörperphysik sein eigen nennt. Aber so, wie man ein Auto zu lenken vermag, ohne es bauen können zu müssen, kann man auch einen Computer programmieren, ohne zu wissen, wie er konstruiert ist. Ein *gewisses* Verständnis für das Gerät als solches, also für das, was die Computerfachleute *Hardware* nennen, ist allerdings von Nutzen, wie es ja auch praktisch sein kann, die Funktion eines Getriebes zu verstehen, wenn man die Gänge wechseln will oder sich die Frage vorlegt, wieso dergleichen überhaupt nötig ist. Die Grundlagen des Programmierens haben mit der Hardware aber nicht viel zu tun, was den normalen User angeht.

Maschinen, die Befehle befolgen, gibt es schon lange. Blaise Pascal, der französische Mathematiker und Philosoph, baute 1642 eine Rechenmaschine. Charles Babbage, ein Engländer, konstruierte 1835 eine ehrgeizige "analytische Maschine"; der britische Staat wurde dazu bewogen, in das Gerät zu investieren. Wie andere staatliche Unternehmungen erwies sich auch dieses angesichts der Technologie der damaligen Zeit als nicht durchführbar, aber die Idee selbst war originell und sinnvoll. Der Franzose Joseph-Marie Jacquard entwickelte zu Beginn des 19. Jahrhunderts ein System gelochter Karten, um das von einem Webstuhl hergestellte Stoffgewebe zu steuern. Dampfpfeifenorgeln auf Jahrmärkten arbeiten nach einem ähnlichen Prinzip. Man kann sich sogar ein Orchester von Berufsmusikern als eine Maschine vorstellen, die Musik produziert; die geschriebenen Notenblätter wären demnach das "Programm". Der Vergleich ist gar nicht übel.

Es hat keinen Sinn, zu einem Jacquard-Webstuhl zu gehen, ihm zu erklären: "Bitte, sieben Meter Tweed, blaugrün kariert", und zu erwarten, daß Sinnvolles geschieht. Diese Sprache spricht die Maschine nicht. Sie versteht nur eine Folge gestanzter Löcher, und für Größe, Zahl und Anordnung dieser Löcher gibt es eine ganze Anzahl strenger Regeln. "Muß i denn, muß i denn" in einer Dampfpfeifenorgel läuft nicht bei einem Webstuhl – und selbst wenn das der Fall wäre, käme dabei etwas heraus, das mit der Originalmelodie nicht mehr viel zu tun hätte. Man müßte wissen, *welche* Folge von Löchern erforderlich ist, um im Gewebe das gewünschte Fadenmuster zu erzeugen. Die Anordnung der Löcher ist somit eine Art *codierter* Version des Gewebemusters, wobei der Mechanismus des Webstuhls als Mittler wirkt.

Ebenso sind die von einem Orchester verwendeten Notenblätter eine codierte Version der beabsichtigten Töne, das Orchester dient als Mittler. (Einen kleinen Unterschied gibt es: Orchester arbeiten nicht so präzise wie eine Maschine, und der Dirigent verfügt über eine gewisse Freiheit, die Musik zu interpretieren. Aber der Vergleich kommt der Sache nahe.)

Bei Computern ist es genauso, nur noch in viel stärkerem Maß. Mit jeder Maschine muß man in der Sprache reden, die sie versteht. In der Frühzeit der

Computertechnik hieß das, daß man lange Listen zu tippen hatte, die etwa so aussahen: Ø1ØØØ111Ø1ØØ11Ø1Ø1111111ØØØ1Ø11ØØØØ1Ø1ØØ11 ... was für die Augen nicht sonderlich gut ist. Im "Innersten" *denken* die Computer immer noch so. (Ø bedeutet dabei "kein elektrischer Stromdurchgang", 1 "elektrischer Stromdurchgang".) Diese recht simple Sprache ist der *Maschinencode* des jeweiligen Computers. Wie durch Zauberei kann man damit alles machen, was man will, und im allgemein hat das den Vorteil, daß es sehr schnell läuft ... aber Programme damit zu schreiben, ist nicht so einfach.

Zum Glück braucht man das heutzutage nicht mehr zu tun.

Der Grund: Man kann einem Computer *beibringen*, Befehle in einer anderen als seiner "Muttersprache" entgegenzunehmen. In Wirklichkeit erarbeitet ein fleißiger Programmierer eine Art "Wörterbuch", um die neue Sprache in Maschinencode zu übertragen, und füttert es in den Computer ein (entweder von einer äußeren Quelle oder eingebaut in die Hardware). Das Wörterbuch trägt den Namen *Compiler* oder *Interpreter*, je nach der Arbeitsweise.

Der Spectrum verfügt über einen eingebauten Interpreter für eine Sprache, die BASIC genannt wird (das heißt "Beginners All-purpose Symbolic Instruction Code", also soviel wie "Symbolischer Allzweck-Befehlscode für Anfänger"). Diese Sprache wurde Mitte der sechziger Jahre in den USA entwickelt und ist weit verbreitet. Sie hat den Vorteil, relativ einfach zu sein, und kann als eine Art Kreuzung zwischen Englisch und Schulalgebra gelten. Was den Anfangsprogrammierer betrifft, spricht der Spectrum nur BASIC. (Über die USR-Taste ist jedoch die Z80-Maschinensprache zugänglich – und wenn Sie nicht wissen, *warum* Sie sie benutzen sollten, haben Sie keine Verwendung dafür. Jedenfalls vorerst noch nicht.)

Ein BASIC-Programm

Statt mit der "Grammatik" von BASIC anzufangen, wollen wir uns einmal ein einfaches BASIC-Programm ansehen, um festzustellen, was es leistet und wie es das macht. Der Vorteil davon: Sie können sofort erkennen, wie leicht das ist. Die grammatikalischen Feinheiten kommen später.

```
1Ø  PRINT "Verdoppeln"

2Ø  INPUT x

3Ø  LET y = x + x

4Ø  PRINT x, y

5Ø  STOP
```

Vermutlich erkennen Sie schon auf den ersten Blick, was hier vorgeht. Zunächst sind aber ein paar Punkte zur Form des Ganzen zu beachten:

a Es besteht aus einer Folge von *Zeilen*.

b Jede Zeile ist ein "zulässiger" (das heißt, logisch vernünftiger) BASIC-*Befehl* oder *-Kommando* oder eine BASIC-Anweisung. (Diese drei Ausdrücke bedeuten in der Praxis alle dasselbe.)

c Jede Zeile beginnt mit einer Nummer, die (wenig überraschend) *Zeilennummer* heißt. (Computer verwenden oft Ø für Null, um die Ziffer vom Buchstaben O zu unterscheiden.)

Diesen recht naheliegenden Bemerkungen muß ich ein paar Klarstellungen nachschicken. Einige davon gelten für *alle* BASIC-Interpreter, wie sie bei anderen Computern verwendet werden, manche nur für den Spectrum. Um mich nicht zu überanstrengen, will ich mich bei den Unterscheidungen nicht aufhalten; wenn Sie zu einem größeren Computer aufsteigen, werden Sie bald dahinterkommen.

Zu a) lohnt keine Bemerkung außer der, daß beim Spectrum eine Programmzeile länger sein darf als eine ganze Zeile auf dem Bildschirm – den Computer stört das nicht.

Einzelheiten zu b) hängen von der BASIC-"Grammatik" ab, auf die wir noch kommen.

Der Grund für die Numerierung von Zeilen c) ist der, daß Sie zu Beginn Ihrer Programmiererlaufbahn den Computer auffordern wollen, bestimmte Zeilen eines Programms auszuführen. Ein Befehl "GOTO 73Ø" befiehlt ihm, das zu tun, was in Zeile 73Ø steht . . . aber er muß wissen, was für eine Zeile das ist. Es hat Vorteile, *nicht* einfach "1, 2, 3 . . ." zu zählen, wie wir noch sehen werden. Die Zeilennummer muß eine ganze Zahl zwischen 1 und 9999 sein, beide eingeschlossen.

Arbeitet der Computer ein Programm durch, so geht er von einer Zeile zur nächsten (außer er erhält als Teil des Programms den Befehl, zu einer anderen Stelle zu gehen). Er braucht also diese Zeilennummern, auch wenn der Programmierer sich gar nicht auf sie beziehen möchte. Der Spectrum nimmt Zeilen ohne Nummern nicht an (führt stattdessen aber vielleicht den Befehl für diese Zeile so aus, als sei er ein Taschenrechner). *Vergessen Sie also auf keinen Fall die Zeilennummern!*

Was macht er?

Tippen Sie das obige Programm in Ihren Spectrum. Die vollständigen Feinheiten der Tastatur erkläre ich etwas später; es wird Ihnen sicherlich aufgefallen sein, daß auf jeder Taste enorm viel zu lesen ist. Manche Tasten können tatsächlich acht verschiedene Wirkungen hervorrufen, je nachdem, in welchem "Modus" und "Shift" sie sich befinden! Die grausige Wahrheit müssen Sie aber nicht auf einmal erfahren.

Ich will davon ausgehen, daß Sie den Computer gerade eingeschaltet und die Meldung "© 1982 Sinclair Research Ltd." erhalten haben. (Wenn nicht, drücken Sie NEW oder ziehen kurz den Stromstecker heraus.)

Nehmen wir Zeile 1Ø als Beispiel. Drücken Sie Taste 1 (oberste Reihe links) und denn Ø (oberste Reihe rechts). Sie sehen die Ziffern unten am Bildschirm auftauchen, zusammen mit einem blinkenden [K]. Dazu sagt man *Cursor*. Drücken Sie als nächstes die Taste "P": Es erscheint das ganze Wort PRINT. Das ist ein sehr raffinierter Zug vom Spectrum und erspart viel Zeit. Durch das Drücken einer einzigen Taste können ganze BASIC-Wörter geschrieben werden. (Es wird Ihnen aufgefallen sein, daß das Wort "PRINT" auch auf der Taste P steht.)

Nun zum Anführungszeichen ". Auch das steht auf der Taste P. Aber in *roter* Farbe. Das heißt: Sie müssen die SYMBOL SHIFT-Taste gedrückt halten, während Sie Taste P drücken, um die Anführungszeichen zu erhalten. Lassen Sie die Taste SYMBOL SHIFT jetzt los. Wenn Sie ein großes V haben wollen, halten Sie die Taste CAPS SHIFT gedrückt, während Sie Taste V drücken. Lassen Sie CAPS SHIFT nun los und drücken Sie der Reihe nach die Tasten E,

R, D, O, P, P, E, L, N. Sie werden in Kleinbuchstaben geschrieben – "erdoppeln". Nun wieder die Anführungszeichen: Wie vorher SYMBOL SHIFT und Taste P.

Zeile 1Ø steht jetzt am unteren Bildschirmrand. (Auch aus dem Cursor ist ein blinkendes [L] geworden.) Drücken Sie, um sie in den Programmspeicher zu stellen, die Taste mit der Aufschrift ENTER. Schon geht sie hinauf!

Bei Zeile 2Ø nicht anders: Drücken Sie 2, Ø, I (auf die Taste, wo INPUT steht) und X: danach ENTER. Die Zeilen 3Ø–5Ø sind ähnlich, aber beachten Sie, daß Sie das Zeichen "=" erhalten, wenn Sie SYMBOL SHIFT niederdrücken und auf [L] tippen; das Zeichen "+" durch SYMBOL SHIFT und [K]; das Komma durch SYMBOL SHIFT und N; STOP ist SYMBOL SHIFT plus A. Vergessen Sie nicht, jedesmal ENTER einzugeben, sobald eine Zeile richtig eingetippt ist.

Wenn Sie sich an diese Anweisungen gehalten haben, steht jetzt das ganze Programm aufgelistet oben auf dem Bildschirm. Dort bleibt es auch ewig sitzen, wenn Sie nichts unternehmen. Das kommt daher, daß Computer nicht bloß Befehlen gehorchen, sondern im Grunde strohdumm sind. Selbst wenn ganz klar ist, was Sie von ihnen wollen, müssen Sie es ihnen trotzdem sagen. Drücken Sie also Taste R (was RUN ergibt) und dann ENTER, damit der Computer weiß, daß Sie mit dem Befehl fertig sind.

Auf dem Bildschirm sehen Sie nun RUN [L], wobei das [L] blinkt. Wenn Sie ENTER drücken, verschwindet das wieder.

Quer auf dem Bildschirm erscheint die Meldung "Verdoppeln". Das ist die Antwort des Computers auf Zeile 10: PRINT "Verdoppeln". Das hat er recht schnell gemacht, und jetzt wartet er darauf, daß Sie *Ihren* Teil von Zeile 2Ø leisten: INPUT x. Zur Erinnerung: Links unten auf dem Bildschirm steht ein [L]. Es will von Ihnen erfahren, was x ist.

Dazu geben Sie eine Zahl ein, gefolgt von ENTER. Versuchen Sie es mit

2 (ENTER)

Blitzschnell zeigt der Computer an

2 4

(und, in der linken unteren Ecke, eine Meldung "9 STOP Statement, 5Ø : 1", über die wir uns noch nicht den Kopf zerbrechen wollen – sie bedeutet lediglich, daß der Computer die Aufgabe richtig bewältigt hat.)

Versuchen Sie es noch einmal. Dazu brauchen Sie nur RUN, gefolgt von ENTER, zu drücken, und schon geht es wieder los. Wenn der Computer x anfordert, probieren Sie etwas anderes aus: Nehmen Sie 756.2912 und drücken ENTER, müßten Sie erhalten:

756.2612 1512.5824

Geben Sie noch ein paarmal RUN und probieren Sie verschiedene x-Werte mit INPUT aus. Ganz wahllos, etwa Ø, 1, 2, 3, 4.

Das müßte Sie davon überzeugen, daß, gleichgültig, was Sie als x eingeben, der Computer zwei Zahlen anzeigt: zuerst x und anschließend das Doppelte von x.

Experimentieren Sie. Das muß zu Ergebnissen führen wie:

17	34
21	42
–5	–1∅
6	12

Wie macht er das?

Ich meine hier nicht im Detail, denn da kämen wir auf Hardware, Maschinensprache und ähnliche Komplikationen zu sprechen. Aber wie sollen Sie, der Programmierer, sich den Computer vorstellen, während er Ihr Programm "fährt", wie man in der Fachsprache sagt? Was geht in dem Siliziumchip-Hirnchen vor? Wenn wir uns auf ein bißchen Vermenschlichung einlassen, ist das ungefähr so:

1∅ PRINT "Verdoppeln"	O je, da kommen Befehle für
2∅ INPUT x	mich, die ich mir merken
3∅ LET y = x + x	muß. Mach mich lieber ran.
4∅ PRINT x, y	Bin neugierig, ob er
5∅ STOP	bald fertig ist.
RUN ENTER	Aha. Geht schon los. Wie war die erste Zeile? Aus dem Speicher holen: 1∅ PRINT "Verdoppeln". Ich soll etwas anzeigen. Das ist ein Anführungszeichen "; dann übernehme ich, was folgt . . .
Verdoppeln	
	bis ich zum nächsten " komme, das mir sagt, ich soll mit dem Anzeigen aufhören. Sonst nichts mehr. Weiter zur nächsten Zeile: 2∅ INPUT x. Er wird mir eine Zahl geben, die ich x nennen muß. Ich zeig ihm einen

[L] Cursor, um ihn dran zu erinnern.

Na schön, ich warte.

Arg langsam, diese Menschen.

2 ENTER

Ah, x ist 2. Was nun? 3Ø LET y = x + x. Das heißt, ich muß x zu x addieren, also 2 + 2 berechnen. Und das Resultat y nennen. Also ist y 2 + 2 = 4. Nächster Befehl? 4Ø PRINT x, y. Ich muß x und y anzeigen, nämlich . . .

2 4

Nächste Zeile? 5Ø STOP, Das wäre demnach alles. Abschließen mit einer Meldung . . .

9 STOP Statement, 5Ø : 1

Die Späße brauchen Sie nicht wichtig zu nehmen: Sehen Sie, wie der Computer die Liste der Befehle einfach durchgeht und sie der Reihe nach befolgt? Er "weiß" nicht einmal, worum es bei dem Programm geht. Aber Sie, der Programmierer, wissen es – es verdoppelt Zahlen.

Die Aufgabe des Programmierers ist damit klar. Wenn Sie eine Aufgabe haben, die vom Computer bewältigt werden soll, müssen Sie eine Folge von Befehlen aufstellen, die, sobald sie ausgeführt sind, dieses Ziel erreichen.

Dazu müssen Sie BASIC genauer kennenlernen. Ihr Spectrum wird recht schlaue Dinge tun, wenn Sie wissen, wie man mit ihm reden muß. Natürlich gibt es alle möglichen Verfeinerungen – ein *gutes* Programm muß nicht nur sein Ziel erreichen, sondern auch ökonomisch und klar sein. Mit den Verfeinerungen hat es aber noch Zeit. Die Hauptsache ist, Programme zu schreiben, die auch *funktionieren.*

Fangen wir an.

2 Arithmetik in BASIC

Grundlage aller Mathematik sind Addition, Subtraktion, Multiplikation und Division. Für den Computer gilt dasselbe. Vernünftigerweise beginnt man also mit der Erklärung für Arithmetik in BASIC.

In Wirklichkeit ist das gelogen – es ist Algebra. Ich wollte Sie nur nicht erschrecken. In fast jedem Programm, das Sie einmal schreiben wollen, sind algebraische Rechenoperationen enthalten, und sei es nur für Buchhaltung. Das ist also ein naheliegender Ausgangspunkt.

Wie normale Algebra verwendet auch BASIC Buchstaben für "allgemeine" Zahlen. In der Fachsprache nennt man sie *Variable* – genauer, *numerische* Variable. Das bedeutet aber lediglich, daß sie Zeichen sind wie x, y, z, a, b, c und so weiter, mit denen algebraische Ausdrücke wie x + y − z aufgestellt werden können. Der Computer kann dann, wenn Sie es ihm durch das Programm befehlen, ausrechnen, welche Werte x, y und z annehmen. (Beispiel: Wenn x = 14, y = 3, z = 9, dann ist x + y − z = 14 + 3 − 9 = 8.) Beim Spectrum können Sie für Variable kompliziertere Symbole verwenden als bloße Einzelbuchstaben, aber lange Namen vergeuden Speicherplatz.

Es gibt ein paar kleine, aber wichtige Unterschiede zwischen BASIC-Algebra und gewöhnlicher Algebra. Die Zeichen + (plus), − (minus), und / (geteilt durch) sind die üblichen. (./. können Sie nicht verwenden.) Das Multiplikationszeichen wird jedoch als Sternchen * geschrieben, so daß 5 * 7 für 5 x 7 steht, was 35 ergibt. Der nach oben zeigende Pfeil ↑ bedeutet "zur Potenz erhoben". Beispiel: 2 ↑ 3 bedeutet, was ein Mathematiker als 2^3 schreiben würde, also 2 hoch 3 (oder 2 *kubiert*), mit dem Wert 2 x 2 x 2 = 8.

Sie können Zahlen, Variable und diese arithmetischen Zeichen zu komplizierteren Ausdrücken verbinden. So ist

a * x ↑ 2 + b * x + c

das bekannte $ax^2 + bx + c$ des Mathematikers.

Kapitel 3 der Einführungsbroschüre erklärt diese Dinge und weist auf einen sehr wichtigen Punkt zur Reihenfolge hin, in der das Gerät die Berechnungen ausführt. Nehmen wir an, der Computer weiß beim obigen Ausdruck, daß a = 4, x = 5, b = 3, c = 7. Sie könnten nun meinen, er berechne das Ergebnis folgendermaßen:

a * x = 4 * 5 = 2Ø,

a * x ↑ 2 = 2Ø ↑ 2 = 4ØØ,

a * x ↑ 2 + b = 4ØØ + b = 4Ø3,

a * x ↑ 2 + b * x = 4Ø3 * x = 2Ø15,

a * x ↑ 2 + b * x + c = 2Ø15 + c = 2022.

Das erhalten Sie, wenn Sie von links nach rechts vorgehen. Es entspräche aber nicht dem $ax^2 + bx + c$ des Mathematikers, das hier den Wert

$$4 \times 5^2 + 3 \times 5 + 7 = 100 + 15 + 7 = 122$$

hätte. Was ist also schiefgegangen?

Der springende Punkt: Die gewöhnliche Algebra kennt viele Regeln darüber, welche Rechenoperationen zuerst auszuführen sind. (In der Schule hatte man sogar Merkwörter dafür, na lassen wir das ...) Die Sprache BASIC besitzt ähnliche Regeln. Dort werden *zuerst* alle ↑ ausgerechnet, dann alle * und / und schließlich die + und −. Bestehen Zweifel, werden sie der Reihe nach von links nach rechts erledigt. Ein a * x ↑ 2 + b * x + c wird demnach so berechnet:

Zuerst die ↑ : x ↑ 2 = 5 ↑ 2 = 25.

Dann die * : a * x ↑ 2 = 4 * 25 = 1ØØ

b * x = 3 * 5 = 15.

Nun die +: a * x ↑ 2 + b * x = 1ØØ + 15 = 115

a * x ↑ 2 + b * x + c = 115 + c = 115 + 7 = 122.

Sie sehen selbst, wieviel Ähnlichkeit das mit normaler Algebra hat.

Wie in der Algebra wollen Sie manchmal einen Ausdruck berechnen, der diesen Regeln nicht auf *natürliche* Weise folgt. Die Lösung ist dieselbe: Sie benützen *Klammern* (). Jeder Ausdruck in Klammern wird als Ganzes vorweg ausgeführt. So würde

(a * x) ↑ 2

berechnet werden als (4 * 5) ↑ 2; das ergibt, wenn Sie den Teil in der Klammer vorweg auflösen, (2Ø) ↑ 2, also 400.

Es gibt nur eine Art von Klammer (also nicht wie [] oder {} in der Algebra). Die anderen Klammern stehen zwar auf der Tastatur, können in Rechenausdrücken aber nicht verwendet werden. Sie müssen also besonders vorsichtig sein, wenn Sie Klammern in Klammern setzen und etwa (3 + 5 * (a − b)) ↑ 4 schreiben, wo der Mathematiker $[3 + 5\,(a - b)]^4$ schreiben würde.

Die Hauptsache: Man muß sich darüber im klaren sein, daß das wirklich ganz wie gewöhnliche Algebra ist, nur mit ein wenig fremdartigen Symbolen.

Wie man einer Variablen einen Wert zuteilt

Das geschieht durch die LET-Anweisung (Taste L). Wenn Sie diese gebrauchen, wo sie hingehört, erkennt der kluge Spectrum sie automatisch als LET und nicht als L (durch ein Verfahren, das "automatische Sytaxprüfung" genannt wird, was aber nur bedeutet, daß er ein Auge darauf hat, ob das, was Sie eintippen, Sinn ergibt). Eine Programmzeile von der Art

```
4Ø  LET x = 5
```

sagt dem Computer: Die Variable x erhält den Wert 5 *von jetzt an, und zwar so lange, bis ihm ein anderer Teil des Programms etwas anderes mitteilt*. Probieren Sie folgendes Programm aus:

```
10  LET a = 4
20  LET b = 3
30  LET c = 7
40  LET x = 5
50  PRINT a * x ↑ 2 + b * x + c
```

Wenn in dem Zwergköpfchen alles in Ordnung ist, werden Sie oben auf dem Bildschirm die Lösung 122 auftauchen sehen.

Es gibt einfachere Wege, gerade zu dieser Lösung zu kommen: versuchen Sie

```
10  PRINT 4 * 5 ↑ 2 + 3 * 5 + 7
```

(Sie können sogar die Zeilennummer weglassen, siehe dazu Kapitel 3 der Einführungsbroschüre. Sie sollen aber sehen, wie LET funktioniert.)

Die rechte Seite eines LET-Befehls kann ein Ausdruck sein, von dem der Computer schon weiß, wie er ihn berechnen muß. So könnten Sie Zeile 50 des obigen Programms abändern, um eine neue Variable y zu definieren, die den Wert a * x ↑ 2 + b * x + c hat:

```
50  LET y = a * x ↑ 2 + b* x + c
```

Fügen Sie

```
60  PRINT y
```

an, dann können Sie überprüfen, ob das richtig funktioniert.

Beispiel: Fallende Körper

Laut Galilei fällt ein Körper aus dem Ruhezustand unter dem Einfluß der Schwerkraft in einer Zeit von t Sekunden (rund) $5t^2$ Meter. Um festzustellen, wie weit er in 17 Sekunden fällt, können Sie das Programm

```
10  LET t = 17
20  PRINT 5 * t ↑ 2
```

verwenden. Als Ergebnis müßten Sie 1445 (Meter) erhalten.

Wollen Sie die Lösung für einen anderen Zeitraum haben, so können Sie die erste Programmzeile verändern. Versuchen Sie es. Wie weit fällt der Körper in:

a 97 Sekunden?

b 3 Sekunden?

c 24.5779 Sekunden?

d 1ØØØØ1 Sekunden?

Eine Verbesserung: Die INPUT-Anweisung

Sie werden es vermutlich satt haben, die Zeile 1Ø immer wieder zu verändern. Ein zivilisiertes Programm würde den INPUT-Befehl nutzen, durch den der Computer Sie fragen kann, welchen Wert eine Variable annehmen soll. Tippen Sie

```
1Ø  INPUT t
2Ø  PRINT 5 * t ↑ 2
```

und geben Sie RUN. Es erscheint der [L]-Cursor, das heißt, der Computer wartet darauf, mitgeteilt zu bekommen, welchen Wert t hat. Geben Sie 17 ein und dann ENTER; Sie erhalten die Antwort, die wir vorher schon hatten. Um aber nun a) bis d) beantworten zu können, brauchen Sie nur wieder RUN zu drücken, müssen aber den neuen Wert von t mitteilen. Probieren Sie das aus.
Weiteres zu INPUT siehe Input/Output, Kapitel 5.

Programmieraufgaben

Hier drei Programme, die Sie schreiben sollen. Verwendet werden nur die Befehle, mit denen wir uns bislang befaßt haben. Es handelt sich um kleine Abwandlungen des Programms für fallende Körper. Wenn Sie hängenbleiben – am Ende des Kapitels finden Sie einen hilfreichen Tip!

e Das Volumen eines Würfels von der Seite x wird bestimmt durch x^3 (das heißt, x ↑ 3 in BASIC, wenn x die Seite bezeichnet). Schreiben Sie ein Programm, mit dem x durch INPUT eingegeben und das Volumen des jeweiligen Würfels mit PRINT angezeigt wird.

f Ein Eimer hängt an einem Seil, das um eine Winde von der Form eines Zylinders mit Radius r gewickelt ist. Für diesen Eimer wird nach den Lehrbüchern der Mechanik die Abwärtsbeschleunigung angegeben mit

$$a = \frac{32}{1 + r^2}$$

Zerbrechen Sie sich nicht den Kopf wegen der Mechanik, sondern schreiben Sie ein Programm, mit dem Sie durch INPUT r eingeben und mit PRINT die Beschleunigung a anzeigen können.

g Im ZEDEX-Spectramarkt kosten ein Glas Honig DM 1.61, eine Packung Zxli-Würfel 50 Pfennig, eine Dose Waskas-Superfleisch 2 Mark. Schreiben Sie ein Programm, das mit PRINT den Gesamtpreis von h Gläsern

Honig, z Packungen Zxli-Würfeln und w Dosen Waskas-Superfleisch anzeigt, wenn Sie h, z und w eingeben. Für das, was ich Ihnen bisher gesagt habe, brauchen Sie drei INPUT-Befehle.

Ein Vorgeschmack von Schleifen

Mit den Befehlen FOR und NEXT können wir ein paar von den bisher genannten Ideen ausprobieren, ohne uns allzuviel Arbeit zu machen. (In vielen Kreisen wird die Vermeidung von Arbeit als "Faulheit" bezeichnet, Programmierer besitzen normalen Sterblichen gegenüber aber einen Vorteil – sie können von "effizienter Programmierung" sprechen und zum Beweis auf Ersparnisse an Speicherplatz oder Zeit verweisen.) Über FOR/NEXT-Schleifen läßt sich vieles sagen, und einige Seiten später werden wir das auch in angemessener Länge tun. Fahren Sie zuerst dieses Programm, damit Sie sehen, was man damit machen kann.

```
10  FOR x = 1 TO 20
20  PRINT x, x * x
30  NEXT x
```

Sie werden eine Liste der Zahlen von 1 bis 20 und eine entsprechende Liste ihrer Quadrate 1, 4, 9 . . . 361, 400 finden. Die FOR/NEXT-Befehle schicken dem Computer nämlich immer wieder durch die Folge von Befehlen zwischen ihnen (hier ist es nur Zeile 20) und setzt die Variable x der Reihe nach auf 1, 2, 3 . . ., bis 20 erreicht ist. Zeile 10 setzt diese Grenzen und löst die Schleife aus. Zeile 30 schickt den Computer erneut durch die Schleife.

Wenn Sie Zeile 20 dieses kleinen Programms verändern, können Sie alles Mögliche in Tabellenform bringen. Kubikwerte gefällig? Versuchen Sie es mit

```
20  PRINT x, x ↑ 3
```

Verändern Sie Zeile 20 der Reihe nach, wie unten angegeben, und beachten Sie die Unterschiede. Die sehr kleinen Veränderungen von einer Zeile zur nächsten sorgen dafür, daß der Computer die Algebra in unterschiedlicher Reihenfolge mit unterschiedlichen Ergebnissen bewältigt. Was berechnen die Programme in üblicher Algebra-Symbolik?

```
h   20  PRINT x, 1/x + 1
i   20  PRINT x, 1/ (x + 1)
j   20  PRINT x, 1/1 + x
k   20  PRINT x, 1/ (1 + x)
l   20  PRINT x, 1 + 1 / x
m   20  PRINT x, (1 + 1) / x
```

Beachten Sie schließlich noch, daß in diesen Programmen die Werte der Variable nicht durch den Befehl LET zugeteilt werden. Stattdessen erhält der Computer die Zuteilung von der FOR-Anweisung.

Lösungen

Fallende Körper

a 47Ø45   b 45   c 3Ø2Ø.3658

d 5ØØØ1ØØØØØØØ. Lassen Sie sich von der scheinbaren Exaktheit nicht täuschen. Der Computer bietet keine Genauigkeit von zwölf Stellen. Er rundet die richtige Lösung ab, die 5ØØØ1ØØØØØ5 lautet.

Programmieraufgaben

e
```
1Ø  INPUT x
2Ø  PRINT x ↑ 3
```

f
```
1Ø  INPUT r
2Ø  PRINT 32 / (1 + 3 * r ↑ 2)
```

g
```
1Ø  INPUT h
2Ø  INPUT z
3Ø  INPUT w
4Ø  PRINT 61 * h + 25 * z + 32.5 * w
```

Ihr Programm braucht nicht *genau so* auszusehen, um "richtig" zu sein. Vor allem ist es allein Ihre Sache, welche Symbole Sie für Variable wählen. Sie können auch die Algebra anders gestalten; bei e) ist nichts einzuwenden gegen 2Ø PRINT x * x * x, und bei f) nichts gegen 20 PRINT 32 / (1 + 3 * r * r).

Schleifen

h $\frac{1}{x} + 1$

i $\frac{1}{x+1} + 1$

j 1 +x (zuerst 1/1, also 1, dann wird x addiert.)

k wie i)

l wie h)

m $\frac{2}{x}$

3 DIE TASTATUR

Jede Taste des Spectrum kann mehrere verschiedene Wirkungen hervorrufen. Hier eine kurze Einführung in die Tastatur.

Wie vorhin schon erwähnt, ist die Spectrum-Tastatur oder das Keyboard sehr verfeinert, und was herauskommt, wenn man eine Taste drückt, hängt auf recht komplizierte Weise vom jeweiligen Zusammenhang ab. Sie werden sich bald daran gewöhnt haben, müssen aber einige Zeit dafür aufwenden, sich auf der Tastatur zurechtzufinden.

Zweierlei beeinflußt die Folgen eines Tastendrucks: Einmal die *Schaltung* (Shift), die benützt wird, und zum anderen die Betriebsart oder der *Modus*, in dem der Computer sich befindet.

Schaltungen

Es gibt zwei Tasten mit den Aufschriften CAPS SHIFT und SYMBOL SHIFT. Hält man eine davon niedergedrückt und drückt gleichzeitig eine andere Taste, wird die Wirkung der letztgenannten Taste verändert. Exakt in welcher Weise, hängt vom Modus ab. Das wollen wir uns näher ansehen:

Betriebsarten

Der Modus ist der "innere Zustand" des Computers und wird angezeigt durch den Cursor. Es gibt fünf Betriebsarten mit den folgenden (blinkenden) Cursors:

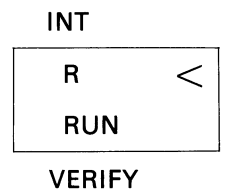
K Keyword mode = Schlüsselwort (Kennwort)-Modus
L Letter mode = Buchstaben-Modus
C Capitals mode = Großbuchstaben-Modus
E Extended mode = erweiterter Modus
G Graphics mode = Grafik-Modus

Erreichen kann man sie auf folgende Weise:

1 Normalerweise befindet sich der Computer im "L"-Modus.
2 Nach einer Zeilennummer oder zu Beginn eines direkt eingegebenen Befehls geht er automatisch in den "K"-Modus über.
3 Um vom "L"- in den "C"-Modus zu gelangen, drücken Sie auf CAPS SHIFT und halten die Taste fest. Geben Sie sie frei, wenn Sie nach "L" zurückwollen. (Das "C" sehen Sie nur, wenn Sie CAPS LOCK verwenden – übergehen Sie das zunächst.)
4 Um vom "L"- in den "E"-Modus zu kommen, müssen Sie *zugleich* CAPS SHIFT und SYMBOL SHIFT drücken. Wiederholen Sie, um nach "L" zurückzugelangen.
5 Um vom "L"- in den "G"-Modus überzugehen, drücken Sie gleichzeitig CAPS SHIFT und Taste 9 (GRAPHICS). Wiederholen Sie, wenn Sie nach "L" zurückwollen.

Wie man was anzeigt

Sehen wir uns irgendeine Taste in den unteren drei Reihen an, meinetwegen Taste "R". Sie sieht so aus:



Das INT ist grün, das < und VERIFY rot.

1 Wenn Sie "r" eingeben wollen, drücken Sie diese Taste im Modus "L".
2 Für "R" drücken Sie CAPS SHIFT und gehen in den "C"-Modus; niederhalten und die Taste drücken.
3 Für "RUN" drücken Sie die Taste im Modus "K".
4 Für "<" müssen Sie in den Modus "L"; halten Sie SYMBOL SHIFT gedrückt und tippen Sie auf die Taste.
5 Für "INT" gehen sie in den erweiterten Modus "E"; beide Shifttasten werden gleichzeitig gedrückt. *Achten Sie auf den "E"-Cursor*: Wenn Sie die Tasten zu lange gedrückt halten, schaltet er möglicherweise wieder nach "L" zurück. Lassen Sie die Shifttasten los und drücken Sie die Taste.
6 Um "VERIFY" einzugeben, gehen Sie wie bei 5) in den "E"-Modus und *halten dann SYMBOL SHIFT niedergedrückt*, während Sie auf die Taste tippen.
7 Um das vom User wählbare Grafikzeichen zu erhalten, das "R" entspricht (siehe das Kapitel über User-wählbare Zeichen), gehen Sie in den Modus "G" und drücken dann die Taste.

Die oberste Tastenreihe

Diese Tasten sind ein bißchen anders. Bei ihnen fehlt die Beschriftung mit Schlüsselwörtern (Modus "K"); stattdessen sieht man ein Grafikzeichen in Kästchenform. Über den Tasten steht in grüner Farbe nichts, dafür gibt es Befehle in weißer Schrift. Das Hauptsymbol auf der Taste und die beiden roten Symbole funktionieren genau wie in den Reihen 2 bis 4. Bei den anderen geht es so:

1 Um das weiße Symbol über der Taste zu erreichen, drücken Sie CAPS SHIFT. Es handelt sich um Steuerzeichen, die selbst *nicht* angezeigt werden.
2 Um das Grafiksymbol zu erreichen, gehen Sie in den "G"-Modus und drücken Sie die Taste.
3 Um das Grafikzeichen in Video-Inversion, also Negativschrift zu erhalten (wechselnd schwarz und weiß oder, allgemeiner, INK und PAPER), gehen Sie in den "G-"Modus, drücken CAPS SHIFT nieder und tippen auf die Taste.

4 Die Farben sind nur Merkhilfen. Beispielsweise ist der Code für GREEN (Grün) 4.
5 Tatsächlich können Sie sogar noch ein bißchen mehr machen; siehe Handbuch, Kapitel 16. Ich habe auch noch nicht erwähnt, was CAPS LOCK, TRUE VIDEO und INVERSE VIDEO bewirken. Statt die Verwirrung noch mehr zu vergrößern, überlasse ich das Experimentieren mit diesen Tasten Ihnen, weil Sie die nicht wirklich *brauchen.*

Fertigprogramme

Am Ende dieses Buches finden Sie eine Anzahl von Programmlistings, die auf Seite 135 beginnen. *Sie können jederzeit eines dieser Listings eingeben und das Programm fahren*, ob Sie die Befehle dort nun verstanden haben oder nicht. Sie brauchen die Zeilen nur über das Keyboard einzutippen, sorgfältig darauf zu achten, daß keine Fehler gemacht worden sind, und RUN zu drücken, gefolgt von ENTER. Wir hoffen jedoch, daß Sie bis zum Schluß des Buches verstanden haben, wie diese Programme funktionieren; damit man aber rasch selbstsicher wird, ist der Versuch sinnvoll, anderer Leute Programme zu fahren. So kann man sich auch eine Vorstellung davon machen, was mit dem Spectrum alles möglich ist.

Die Programme sind in der Regel von Programmhinweisen begleitet, die verschiedene Eigenheiten erklären, hier und dort Veränderungen des Programms vorschlagen oder auf verwandten Gebieten Aufgaben für Sie bereithalten. Die Programme laufen aber, falls keine Eingabefehler gemacht worden sind, ob Sie die Hinweise ganz verstanden haben oder nicht. Die meisten Programme veranschaulichen bestimmte Methoden, die in diesem Buch erklärt werden, aber manche nutzen auch Ideen, für deren genaue Erläuterung mir der Platz fehlt. Auf jeden Fall gehe ich davon aus, daß Sie nicht nur diesen Band hier, sondern auch das Handbuch von Sinclair lesen.

4 HIIIIIIILFEEEEEE!

Läuft nicht so gut, wie Sie gehofft hatten? Vielleicht brauchen Sie Hilfe.

Was tut man, wenn etwas nicht richtig klappt?
Im allerschlimmsten Fall ziehen Sie kurz den Stromstecker heraus. Dann bricht das Programm zusammen und alles, was Sie eingegeben haben, wird gelöscht – gleichzeitig sind aber auch alle Schwierigkeiten behoben (es sei denn, sie lägen in der Hardware, dann hätten Sie Pech gehabt). In der Regel gibt es aber bessere Methoden.

a Wenn Sie in einer Programmzeile etwas Falsches eingeben: Verwenden Sie die mit Pfeilen bezeichneten Tasten 5 und 8 (denken Sie daran, daß Sie CAPS SHIFT brauchen), um den [L]-Cursor knapp an dem Störenfried vorbeizuführen, und löschen Sie mit DELETE (Taste Ø mit CAPS SHIFT).

b Wenn Sie eine Zeile verändern wollen, die schon im Programm steht:
Um sie ganz zu löschen, schreiben Sie ihre Zeilennummer und drücken ENTER.
Im anderen Fall führen Sie den > -Cursor mit 6 und 7 an den Zeilen hinauf und hinunter und drücken anschließend EDIT, um die Zeile dort hinunterzuholen, wo daran gearbeitet werden kann. Fahren Sie dann fort wie unter a).
Diverse Kniffe: Es macht arge Mühe, den Cursor Zeile für Zeile von Zeile 1Ø bis Zeile 53Ø herunterzuführen. Finden Sie eine Zeilennummer kurz vor 53Ø, die Sie nicht verwendet haben, etwa 529. Geben Sie 529 ENTER ein und drücken Sie EDIT. Sie werden sehen, daß Zeile 53Ø herunterkommt. (Warum tut sie das?)

c Wenn das Programm steckenbleibt und sich nichts zu rühren scheint:
Das kommt während eines Debuggingdurchgangs oft vor. Das Letzte, was Sie wollen, wäre, alles zu löschen und wieder neu einzugeben; außerdem spricht viel dafür, daß doch wieder dasselbe eintritt.

c1 Wenn das Programm noch läuft, Sie es aber anhalten wollen: Drücken Sie CAPS SHIFT und BREAK.

c2 Wenn es bei einem numerischen INPUT-Befehl steckenbleibt: BREAK funktioniert zwar nicht, aber STOP. Wenn Sie aus Versehen dort schon etwas stehen haben, erhalten Sie ständig einen [?]-Cursor für einen Syntaxfehler, eine in hohem Maß ärgerliche Sache. Löschen Sie das falsche Zeug mit DELETE und drücken Sie anschließend STOP.

c3 Wenn es bei einer Zeicheneingabe hängt – [L]-Cursor: drücken Sie DELETE und *dann* STOP.

d Wenn das Programm zwar läuft, aber nicht das Richtige tut: Lesen Sie die Kapitel über DEBUGGING (Fehlersuche).

5 INPUT/OUTPUT

Es gibt viele nützliche Wege, Information in einen Computer einzugeben oder sie herauszuholen und gut organisierte Displays zu produzieren: Input/Output.

Am INPUT-Befehl ist viel mehr dran, als ich bisher verraten habe. So können Sie auf einmal mit INPUT mehrere Zahlen eingeben. Ein besserer Weg, die Programmieraufgabe g) in Kapitel 2 zu lösen, ist dieser:

```
10  INPUT h, z, w
20  PRINT 61 * h + 25 * z + 32.5 * w
```

Wenn Sie das mit RUN fahren, werden Sie feststellen, daß Sie nach *jeder* der drei INPUT-Eingaben ENTER drücken müssen. Die Zahlen werden vorübergehend in der untersten Reihe (oder den Reihen) angezeigt, bis alle drei eingegeben sind.

Die Beistriche, die sie trennen, steuern außerdem die Stellen, wo die Zahlen angezeigt werden: Sie gehen abwechselnd in die Spalten Ø und 16. Wenn Sie statt der Beistriche Strichpunkte ";" nehmen, sehen Sie, daß die Eingaben der Reihe nach ohne Zwischenräume angezeigt werden. Um Zwischenräume zu erhalten, müssen Sie diese im INPUT-Befehl definieren:

```
1Ø  INPUT h; "□"; z; "□"; w
```

wo ein Kästchen □ einen Zwischenraum anzeigt.

Sie können mit einer Eingabe auch *Aufforderungszeichen* anzeigen: Hinweise, die Sie daran erinnern, welche Eingabe erforderlich ist. Dazu führen Sie den Hinweis in Anführungszeichen als Teil der INPUT-Anweisung auf. Beispielsweise verändern Sie Zeile 1Ø oben zu:

```
1Ø  INPUT "Honig", h
12  INPUT "Zxli Würfel", z
14  INPUT "Waskas", w
```

Sie können als Teil einer einzelnen INPUT-Anweisung kompliziertere Kombinationen solcher Befehle verwenden; was gemeint ist, läßt sich aber erkennen.

Output

Wenn wir bisher Output- (also Ausgabe)-Daten, etwa Zahlen, mit PRINT anzeigten, haben wir dem Computer die Entscheidung überlassen, *wo* er sie anzeigen wollte. Nun ist das nicht immer praktisch; es führt auch nicht immer zu schönen Displays. Wenn Sie die Anzeigeposition verändern wollen, verwenden Sie PRINT AT.

Für Anzeigezwecke stellt man sich das Bildschirm-Display aufgeteilt in 22 *Reihen* von oben nach unten mit den Nummern Ø–21 und 32 *Spalten* von links nach rechts mit den Nummern Ø–31 vor. Im späteren Kapitel GRAFIK veranschaulicht das ein Diagramm, hier zunächst aber ein Programm, mit dem Sie experimentieren können.

```
1Ø  INPUT "Reihe", r
2Ø  INPUT "Spalte", s
3Ø  PRINT AT r, s; "DM"
4Ø  GO TO 1Ø
```

Damit läßt sich buchstäblich Geld drucken! Probieren Sie verschiedene Werte für die Nummern von Reihen und Spalten aus und sehen Sie sich an, wo die Deutschen Mark angezeigt werden.

Um eine bestimmte PRINT-Position auszuwählen, geben Sie im Befehl natürlich r und s als Zahlen an. Wenn Sie eine Nachricht in die unterste Reihe setzen wollen, beginnend nach fünf Leerstellen, (also in Spalte 4, weil die Zahlen mit Ø anfangen!), schreiben Sie

```
1Ø  PRINT AT 21, 4; "Nachricht"
```

Um ungefähr in die Bildschirmmitte zu kommen:

```
1Ø  PRINT AT 10, 12; "Nachricht"
```

Und so weiter und so fort.

Wenn Sie die AT-Anweisung nicht verwenden, geht der Computer automatisch einfach zur "nächsten" Position weiter. Wo das ist, hängt von dem jeweils Angezeigten ab. Wenn der letzte PRINT-Befehl nicht mit ";" oder "," aufhört, geht der Computer weiter zur nächsten Zeile. Ein ";" führt ihn zur nächsten Stelle in *derselben* Reihe, ein "," zum nächsten verfügbaren Platz in Spalte Ø oder 16, je nachdem, welcher als erster frei ist.

Doppelpunkt und Komma haben bei INPUT-Befehlen ähnliche Wirkungen.

In Verbindung mit diesen automatischen Merkmalen ist der PRINT-Befehl dann sehr nützlich, wenn Sie Daten in organisierten Kolonnen schreiben wollen. Beispiel: Mit dem folgenden Programm können Sie ein privates Telefonverzeichnis anlegen.

```
1Ø  INPUT "Name"; n$
2Ø  INPUT "Amt", a$
3Ø  INPUT "Nummer"; t
4Ø  PRINT TAB 1; n$; TAB 15; e$; TAB 25; t
5Ø  GO TO 1Ø
```

(TAB ist Taste P im erweiterten Modus. Die Dollarzeichen stehen für *Strings* und werden später in einem eigenen Kapitel erklärt.) Fahren Sie mit RUN und geben Sie auf Anforderung als INPUT-Daten etwa ein:

Fritz	Timbuktu	44399
Karl	Basel	12345
Polizei	München	11∅

Sie sehen, wie die Daten säuberlich in drei Kolonnen angeordnet werden. (Zum Anhalten drücken Sie bei einer Zahleneingabe STOP, bei einer Zeichenstring-eingabe zuerst DELETE und dann STOP.)

Aufgabe

Schreiben Sie ein Programm, mit dem Sie über INPUT 22 Zahlen eingeben können. Zeigen Sie sie mit PRINT in diagonaler Folge von oben links nach unten rechts an (Sie verwenden dabei etwas in der Art von PRINT AT i, i; n). Überlegen Sie dann, wie das von oben rechts nach unten links geht (PRINT AT i, 21–i).

6 SCHLEIFEN

Zehne lagen im Bettchen, und der kleinste rief:
"Rollt alle rüber, einer liegt schief!"
Alle rollten rüber, einer fiel hinaus.
Da war es noch nicht aus.
Neune lagen im Bettchen, und . . .

Mit den Befehlen FOR und NEXT können Sie den Computer veranlassen, einen Befehl so oft zu wiederholen, wie Sie das verlangen. Das klingt vielleicht nicht maßlos interessant, aber in Wahrheit ist das eine der nützlichsten Waffen im Arsenal des Programmierers. Sie können nämlich durch Variable verändern, was jeder Schritt bewirkt.

Wir haben schon gesehen, wie wir mit FOR/NEXT-Schleifen Tabellen von Werten einer Funktion wie x ↑ 3 anzeigen können. Hier eine nicht ganz so simple Anwendungsart.

Die Fakultätsfunktion n! wird, wie jeder Schüler weiß, definiert durch:

$$n! = n\,(n-1)\,(n-2)\,(n-3)\ldots 3 \times 2 \times 1.$$

Sie erhalten die Zahl der Möglichkeiten, n Gegenstände der Reihenfolge nach zu ordnen. Beispiel: 3! = 3 x 2 x 1 = 6; und die Buchstaben abc können auf sechsfache Weise angeordnet werden: abc, acb, bac, bca, cab, cba.

Zur Berechnung von n! können wir Schleifen verwenden. Dabei geht es darum, in Stufen zu rechnen: 1; 1 x 2; 1 x 2 x 3; 1 x 2 x 3 x 4; . . . und zwar so lange, bis die größte Zahl n ist. Jedesmal nehmen wir das Resultat des *vorherigen* Schritts und multiplizieren mit der nächsthöheren Zahl. Das ist genau der Vorgang, den man bei einer Schleife braucht. Die Folge von Anweisungen

```
1Ø  LET i = 1
2Ø  FOR k = 2 TO n
3Ø  LET i = i * k
4Ø  NEXT k
```

hat denn auch dieselbe Wirkung.

Was bewirkt das Programm? Und wie macht es das? Zeile 10 setzt einen Startwert für die Variable i fest. Zeile 20 weist den Computer an, nachfolgende Zeilen immer wieder auszuführen und k der Reihe nach die Werte 2, 3, 4, 5 . . ., n zuzuteilen. Zeile 4Ø sagt ihm, wann er das Ende dieser Befehle erreicht hat und wieder zum Anfang der Schleife zurückkehren soll. Nach dem ersten Durchgang nimmt er also k = 2 an, und Zeile 3Ø berechnet

```
i = i * k = 1 x 2
```

Beim nächstenmal ist k zu 3 geworden und i zu 1 x 2, also wird i * k berechnet, was nun 1 x 2 x 3 ergibt. Das nächstemal hat k den Wert 4, gerechnet wird 1 x

2 x 3 x 4. Und das immer so weiter bis zum letzten Schritt, wenn k zu n geworden ist und der Computer i = 1 x 2 x 3 x 4 x . . . x n rechnet. Dank der in Zeile 2Ø festgelegten Grenzen weiß er dann, daß die Schleife beendet ist.

Das berechnet für Sie aber immer noch nicht n!, weil das Programm keine Anweisungen dafür enthält, was n ist, oder dafür, daß das Ergebnis angezeigt werden soll. Sie müssen die Schleife also in ein größeres Programm einbetten:

```
 5  INPUT n
1Ø  LET i = 1
2Ø  FOR k = 2 TO n
3Ø  LET i = i * k
4Ø  NEXT K
5Ø  PRINT "Fakultät □"; n; "□ ist □"; i
```

(Zeile 5Ø dient nur für eine schönere Anzeige; Sie erhalten Ausgaben wie

Fakultät 6 ist 72Ø

Beachten Sie die als □ dargestellten Leerräume, damit es hübsch aussieht.)

Wie Kaninchen sich vermehren

Um 122Ø kam ein gewisser Leonardo von Pisa mit dem Spitznamen Fibonacci (Sohn der guten Laune) auf ein interessantes Problem in Zusammenhang mit Kaninchen.

Wenn ein Kaninchenpaar einmal im Monat ein Paar Nachkommen hervorbringt und es einen Monat dauert, bis die Nachkommen fruchtbar werden, und wenn Sie mit einem Paar anfangen – wie sieht das Wachstum dann aus? Der Einfachheit halber gehen wir davon aus, daß die Produktion jeden Monat gleich ist, und daß jedes Paar aus einem Männchen und einem Weibchen besteht.

Eine Tabelle kann hier von Nutzen sein:

Monat	Zahl der prod. Paare	Zahl der neuen Paare

Im Monat Ø haben wir 1 produzierendes Paar, also Eintrag

Ø   1   Ø

Im Monat 1 erhalten wir nur 1 neues Paar:

1   1   1

Im Monat 2 erhalten wir 1 neues Paar, und das vorherige neue Paar wird produktiv:

2   2   1

Monat	Zahl der prod. Paare	Zahl der neuen Paare

Im Monat 3 sind sie alle produktiv, und wir erhalten 2 neue Paare:

3	3	2

und so weiter:

4	5	3
5	8	5
6	13	8
7	21	13

Wir kriegen also eine ganze Menge Kaninchen zusammen, was ja nicht weiter verwunderlich ist.

Soll die Gesamtzahl der Kaninchenpaare im Monat m also f (m) sein, so haben wir f (∅) = 1, f (1) = 2, f (2) = 3, f (3) = 5, f (4) = 8, und so weiter.

Gehen wir allgemeiner vor. Nehmen wir an, im Monat m hätten wir p produzierende Paare und n neue Paare. Im nächsten Monat werden sie alle produktiv, was p + n *produzierende* Paare ergibt; die p produzierende Paare liefern uns p *neue* Paare. Die Tabelle hat also zwei aufeinanderfolgende Zeilen, die so aussehen:

Monat	Zahl der prod. Paare	Zahl der neuen Paare
m	p	n
m+1	p+n	p

Diese Tabelle läßt erkennen, wie wir mit Hilfe von Schleifen f (m) berechnen können. Es geht darum, daß wir, um vom Monat m zum Monat m + 1 zu kommen, das alte p in p + n und das alte n in p verwandeln müssen. Folgendes funktioniert:

```
1∅  LET p = 1
2∅  LET n = ∅
3∅  INPUT m
4∅  FOR t = 1 TO m
5∅  LET c = p
6∅  LET p = p + n
7∅  LET n = c
8∅  NEXT t
9∅  PRINT "f ("; m;") □ ist □"; p + n
```

Wenn Sie sich näher damit befassen, werden Sie sehen, daß das genau den Schritten entspricht, mit denen die Tabelle aufgebaut wurde. Die Zeilen 1Ø und 2Ø setzen die Ursprungswerte für p und n. Zeile 3Ø will wissen, für welchen Wert von m wir f (m) wollen. Die Zeilen 4Ø bis 8Ø durchlaufen eine Schleife, die aufeinanderfolgende Zeilen der Tabelle generiert. Beachten Sie Zeile 5Ø, die den *alten* Wert von p für die Verwendung in Zeile 7Ø in Erinnerung behält und ihn c nennt. (Wenn Sie das nicht täten, würde Zeile 6Ø p zu früh verändern, und Zeile 7Ø würde den falschen Wert für n liefern.)

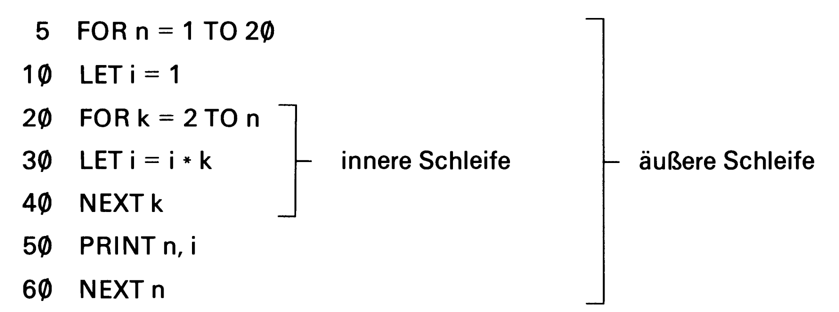
Die Zahlen f (m) werden *Fibonacci-Zahlen* genannt. Wenn Sie bemerken, daß die Spalten p und n in der Tabelle dieselben Zahlen enthalten, nur um eine Zeile verschoben (warum?), werden Sie sehen, daß $f(m + 2) = f(m + 1) + f(m)$, das heißt, jede Fibonacci-Zahl ist die Summe der beiden vorangegangenen.

Was ist der Wert von f (14)? Und der von f (77)?

Verschachtelte Schleifen

Es kommt noch besser: Sie können Schleifen in Schleifen oder sogar Schleifen innerhalb von Schleifen in Schleifen setzen (falls der Speicherplatz reicht). Sie würden staunen, wenn Sie wüßten, wie oft das notwendig ist.

Beispiel: Nehmen wir an, Sie wollen eine Tabelle von Werten der Fakultätsfunktion anzeigen. Sie können eine Schleife nehmen, wie sie auf Seite 17 für x ↑ 3 steht, aber dann brauchen Sie eine *zweite* Schleife für die Berechnung von n!. Sie erhalten also etwas in der folgenden Art:



Beachten Sie, daß die Schleife "FOR n/NEXT n" völlig außerhalb der von "FOR k/NEXT k" steht. Das muß sein, damit das überhaupt Sinn ergibt – genau wie bei Klammern. Ein Ausdruck $a + b (b - 2c) + d$ ergibt sehr wohl Sinn, nicht aber $[a + (b - 2c]) + d$. Vertauschen Sie einmal die Zeilen 4Ø und 6Ø und sehen Sie sich an, was dabei herauskommt. Bringt nicht viel, wie? Der Haken: Der Spectrum akzeptiert und fährt zwar Programme mit falsch verschachtelten Schleifen, aber die Ergebnisse entsprechen natürlich nicht dem, was Sie gewollt haben. Er bringt *keine* Fehlermeldung, also Vorsicht!

Schrittgröße

Wenn Sie einfach FOR i = 1 TO 2Ø schreiben, *unterstellt* der Computer, daß die Variable i die Werte 1, 2, 3, 4, 5, . . ., 19, 2Ø annehmen soll. Das heißt, er geht davon aus, daß Sie in *Schritten* der Größe 1 vorgehen.

Das brauchen Sie aber nicht zu tun. Sie können eine andere Schrittgröße dadurch setzen, daß Sie die Taste STEP verwenden. Beispielsweise läßt die Anweisung

```
1Ø   FOR j = −3 TO 3 STEP .5
```

j die Werte −3, −2.5, −2, −1.5, −1, −Ø.5, Ø, Ø.5, 1, 1.5, 2, 2.5, 3 durchlaufen. In gleicher Weise durchläuft

```
1Ø   FOR j = 3 TO 4 STEP .01
```

die Werte 3, 3.Ø1, 3.Ø2, 3.Ø3 . . . in Hundertstelschritten, bis es mit 3.98, 3.99, 4 aufhört.

Sie können mit STEP auch abwärtsgehen:

```
FOR j = 1Ø TO Ø STEP −1
```

läßt j die Folge 1Ø, 9, 8, 7, 6, 5, 4, 3, 2, 1, Ø durchlaufen. Erinnert Sie das an etwas? Fahren Sie dieses Programm:

```
1Ø   FOR j = 1Ø TO Ø STEP −1
2Ø   PRINT j
3Ø   NEXT j
4Ø   PRINT "ABHEBEN!"
```

Nicht gerade das Neueste, aber immerhin . . .

7 DEBUGGING I

Es heißt, der anschauliche Ausdruck "die Bugs (Insekten) herausholen" sei in der Anfangszeit der Computerentwicklung entstanden, als Insekten in die Maschinen krochen und Kurzschlüsse verursachten. Wenn heutzutage ein Computer versagt, liegt es meistens am Programmierer. Um das zu beheben, müssen Sie trotzdem etwas verstehen von Debugging.

Wie alle Programmierer sehr bald erfahren, ist es eine harte Tatsache des Lebens, daß Programme kaum jemals funktionieren – jedenfalls nicht beim ersten Lauf. Das Verfahren, die Fehler in einem Programm zu beseitigen, nennt man Debugging, zu deutsch Fehlersuche oder F-Beseitigung. Es ist wichtig, systematisch an diese Sache heranzugehen, weil sogar bei einem kleinen Programm die Fehlerquelle nicht unbedingt leicht zu finden ist. Und es gibt nur wenig, was ärgerlicher sein kann als Programme mit Fehlern, die sich nicht finden lassen.

Sehen wir uns zunächst einmal die Fehlerarten an, die auftreten können. Grob gesprochen, zerfallen sie in zwei Gruppen: *Syntax*fehler und *Ablauf*fehler.

Syntaxfehler

Das sind Fehler, die der Computer sofort erkennen kann, wenn Sie sie eingegeben haben. Beispiel: Angenommen, ich tippe die Zeile

```
FOR p = 1 - 7
```

in der irrigen Meinung, das Symbol "–" könne als Entsprechung für "TO" gelten. (Das ist, wenn Sie so wollen, ein Fehler im Gebrauch der Grammatik der Sprache – daher der Ausdruck "Syntax".)

Der Spectrum ist in diesem Fall für Anfänger besonders hilfreich. Er läßt zwar zu, daß Sie das Symbol "–" eingeben, erkennt aber, daß es keine Umstände gibt, unter denen eine vollständige Programmzeile als 5Ø FOR p = 1 – 7 erscheinen kann, zeigt ein [?]-Hinweissymbol (um mitzuteilen, daß ein Syntaxfehler vorliegt) und weigert sich, die Zeile anzunehmen, bis das anstößige "–" ersetzt wird. (Viele weit verbreitete Mikrocomputer erlauben ohne weiteres, daß Sie in ein Programm solche falschen Befehle einstreuen, und erheben erst Einwände, wenn Sie es fahren wollen. Einem erfahrenen Programmierer macht das nichts aus, aber der Anfänger neigt zu solchen Fehlern, und es spart viel Zeit, wenn man sofort auf sie aufmerksam gemacht wird.)

An diesem Punkt mögen manche Leser von einer Frage geplagt werden, die wir so formulieren könnten: "Wenn der Spectrum schon weiß, daß das einzige, was nach der "1" in Anweisung 5Ø erscheinen kann, das Schlüsselwort "TO" ist, warum setzt der Computer das "TO" dann nicht von selbst ein?"

Die Antwort: Soviel, daß er das tun könnte, weiß er nun auch wieder nicht. Auf die "1" könnte beispielsweise eine andere Ziffer folgen, wie in

```
5Ø   FOR p = 12 TO 4Ø
```

Es gibt noch andere, kompliziertere Möglichkeiten. Der Befehl

```
5Ø   FOR p = 1 - 7 TO 5
```

ist zulässig und bedeutet dasselbe wie 5Ø FOR p = – 6 TO 5.

Der [?]Hinweis kann schon auftauchen, bevor Sie eine Zeile fertiggeschrieben haben. Beispiel:

```
2Ø   LET e * s = q
```

Der Computer erwartet nach e ein "="-Symbol – oder einen anderen Buchstaben oder eine Zahl – also stellt er den [?]-Hinweis vor das "*", allerdings erst, nachdem ENTER gedrückt worden ist.

Die Regel bei Syntaxfehlern lautet also: *Achten Sie auf den [?]-Hinweis.* Wenn er erscheint, liegt der Grund meistens ziemlich klar. Wenn nicht, vergleichen Sie die Anweisung, die Sie schreiben, mit den entsprechenden Abschnitten des Handbuchs. (Eine Fehlerquelle, die gelegentlich Verwirrung stiftet, tritt dann auf, wenn Sie versehentlich ein Schlüsselwort ausschreiben, statt die Spezialtaste dafür zu benützen. Beispiel: Wenn Sie statt des Schlüsselworts "TO" die beiden Buchstaben "T" und "O" eintippen. Der Computer nimmt das nicht an, obwohl auf dem Bildschirm kein Unterschied zu erkennen ist.)

Ablauffehler

Es gibt viele Arten von Fehlern, die auftreten können, wenn ein Programm gefahren wird. Ich hatte einmal mit einer Computerroutine zu tun, die wegen eines recht versteckten Fehlers nur bei Programmen mit einer geraden Zahl von Zeichen funktionierte; bevor das behoben wurde, konnte man es dadurch umgehen, daß man an einer unschädlichen Stelle bei allem, was nicht lief, ein zusätzliches Zeichen anfügte!

Es ist von größerem Nutzen, konkrete Beispiele möglicher Fehler zu geben, als sie allgemein aufzuführen. Sehen wir uns zunächst einmal den folgenden Codeteil an:

```
1Ø   FOR p = - 5 TO 5
2Ø   LET a = 1Ø / p
3Ø   PRINT a
4Ø   NEXT p
```

Jede Zeile ist absolut zulässiges BASIC, es würden keine Syntaxfehler angezeigt werden. Das Programm läuft auch ganz richtig an, sobald RUN gegeben wird, und zeigt auf dem Bildschirm

−2	d. h. 1Ø / (− 5)
−2.5	d. h. 1Ø / (− 4)
−3.3333333	d. h. 1Ø / (− 3)
−5	d. h. 1Ø / (− 2)
−1Ø	d. h. 1Ø / (− 1)

Dann hält es aber an mit der Fehlermeldung

6 Number too big, 2Ø : 1 (Zahl zu groß)

Hier ist das Problem ziemlich klar – auch ohne ein Nachschlagen der Fehlernummer im Handbuch. Erstens zeigt die Meldung, daß der Computer Einspruch gegen Zeile 20 erhebt, die da lautet

```
2Ø   LET a = 1Ø/p
```

Zweitens ist diese Anweisung schon mehrmals ohne Widerspruch ausgeführt worden. Demnach muß das Problem bei einer der Größen liegen, die sich *verändern*, also der Wert von a oder p. Der Wert von p, der eben erfolgreich behandelt wurde, ist −1, der laufende Wert von p also Ø. Mit anderen Worten: Der Computer versucht 1Ø/Ø zu rechnen, was unendlich ist (oder nicht nach Geschmack definiert) und demnach nicht bewältigt werden kann, gleichgültig, wieviel Speicherplatz das Gerät für die Lösung bereitstellt.

Sie sind zu dieser Schlußfolgerung vermutlich längst gelangt, bevor Sie die (ziemlich weitschweifige) Analyse zu Ende gelesen hatten, aber ich benütze dieses einfache Beispiel dazu, die *Arten* von Hinweisen zu veranschaulichen, auf die Sie achten sollten, wenn Sie mit einem Fehler nicht zu Rande kommen.

1 Finden Sie die anstößige Zeile (die Nummer nach dem Komma in der Schlußmeldung).
2 Stellen Sie fest, ob diese Anweisung mindestens einmal ausgeführt worden ist, bevor die Fehlermeldung entstand. (Wenn ja, liegt das Problem im bestimmten Wert einer der Variablen in dem Augenblick, als der Fehler auftrat.)
3 Nutzen Sie die Meldung, um einen weiteren Hinweis zu erhalten. In unserem Beispiel ist das "Nummer too big". Beachten Sie, daß die Meldung nicht *genau* mitteilt, was geschehen ist, (also nicht etwa "Versuch, durch Null zu teilen"), so daß es nicht immer genügt, sich einfach eine Fehlermeldung anzusehen, in der Hoffnung, sie werde genau erklären, was passiert ist.

Zur Form der Fehlermeldungen des Spectrum sind noch zwei Dinge zu sagen.

Erstens: Sie beginnen mit einer Zahl oder einem Buchstaben (hier 6), wobei es sich nur um ein Kennzeichen handelt, das sich auf einen Eintrag in Anhang B des Handbuchs bezieht (oder Seite 192). Dieser Eintrag hilft Ihnen vielleicht bei der Prüfung, was schiefgegangen ist, weiter – oder auch nicht. In diesem Fall sagt das Handbuch: "Die Berechnungen haben zu einer Zahl geführt, die größer ist als rund 1Ø^{38}", was sich für mich nicht viel anders anhört als "Number too big . . ."

Zweitens: Am Ende der Meldung, nach dem Doppelpunkt, steht eine Zahl, die normalerweise 1 ist. Das bezieht sich auf "Mehrfachbefehl-Zeilen", wo eine einzige Zeile mehr als einen Befehl enthält. Diese Methode haben wir bisher noch nicht verwendet; Einzelheiten dazu finden Sie in Kapitel 9 über "Verzweigungen". Grob gesprochen, geht das so, daß Sie mehrere Befehle in eine einzige Zeile setzen können, getrennt durch Doppelpunkte. Diese letzte Zahl der Meldung teilt Ihnen mit, welcher Befehl der Sünder ist. Also bedeutet 20 : 1 "der erste Befehl in Zeile 2Ø", und 2Ø : 3 würde sich auf den dritten beziehen.

Das scheint sehr vernünftig zu sein (und ist es auch), aber wenn man nicht aufpaßt, kann hier Verwirrung entstehen. Der Grund: Der Spectrum betrachtet jede "natürliche Unterbrechung" in der Folge der Anweisungen (wie etwa THEN) als Beginn einer neuen "Anweisung" für diesen Zweck. Demnach würde die Zeile

```
10  IF p/0 = 2 THEN LET p = p + 1
```

die Meldung

```
6 Number too big, 10 : 1
```

hervorrufen.
Stünde dort aber

```
10  IF p = 2 THEN LET p = p/0
```

dann bekämen Sie

```
6 Number too big, 10 : 2
```

zu sehen, weil der Fehler im zweiten Teil des Befehls steckt.
Übrigens erklärt das die seltsamen Meldungen, die Sie erhalten, wenn *nichts* fehlgeht. Tippen Sie beispielsweise LIST; dann erhalten Sie

```
0  O.K., 0 : 1
```

und das heißt

0	Meldecode 0: Der Computer hat getan, was von ihm verlangt wurde, und ist auf keine Probleme gestoßen.
O.K.	Knappere Fassung des Obigen.
0	Er hat eben "Zeile 0" ausgeführt, was nichts anderes heißt, als daß der Befehl keine Zeilennummer besaß.
:1	Es war der erste Befehl in dieser Zeile.

Das Problem der Teilung durch Null taucht verhältnismäßig oft auf, nicht immer unbedingt in so auffälliger Weise, wie hier dargestellt. Beispiel:

```
30  INPUT p, q, r
40  LET a = (p + q - r * 2) / (5 + (p - r) * (p - r) - 2 * q)
```

Dadurch entsteht dasselbe Problem, wenn die Werte 7, 15 und 2 der Reihe nach für p, q und r eingegeben werden. (Versuchen Sie es einmal!) Im allgemeinen ist es zweckmäßig, Divisoren darauf zu überprüfen, ob sie Null sind, bevor man zu rechnen versucht. Wir könnten das obige Beispiel folgendermaßen umschreiben:

```
30  INPUT p, q, r
32  LET d = 5 (p + r) * (p − r) − 2 * q
34  IF d = 0 THEN PRINT "Geht nicht": GO TO 30
40  LET a = (p + q − r * 2) / d
```

Der Sinn des GOTO-Befehls ist (hoffentlich) klar! Zeile 34 ist ein Beispiel für eine Zeile mit Mehrfachbefehlen.

Debugging-Problem

Hier schließlich noch eine Gelegenheit, das erworbene Wissen zu überprüfen. Das folgende Programm soll eine Reihe positiver Zahlen akzeptieren und ihr Mittel anzeigen. Wenn wir das Mittel von 2.4, 8.1, 7 und 14 erfahren wollten, würden wir eingeben:

```
2.4
8.1
7
14
−1
```

Das "−1" am Ende soll anzeigen, daß keine weiteren Daten eingegeben werden, und gehört *nicht* zu den Daten. (Ein solcher Wert wird oft *Begrenzer* genannt; wenn Sie sich unten Zeile 40 des Programms ansehen, erkennen Sie, wie er wirkt.)

```
 10  LET s = 0
 20  LET c = 0
 30  INPUT n
 40  IF n < 0 THEN GO TO 100
 50  LET c = c + 1
 60  LET z = s + n
 70  GO TO 30
100  PRINT "MITTEL IST □"; s/c
```

Das Listing enthält einige Schreibfehler. Stellen Sie fest, ob Sie sie korrigieren können, indem Sie das Programm so eingeben, wie es dasteht, und daran herumbasteln. Wenn Sie es versucht haben, vergleichen Sie Ihre Lösung mit der meinen auf Seite 53.

Viel Spaß bei der Fehlersuche!

8 ZUFALLSZAHLEN

Eine Spur von Unberechenbarkeit ist in den Spectrum ebenfalls eingebaut.

Die Anweisung RND erzeugt eine "zufällige" Zahl zwischen 0 und 1, die Ø entsprechen kann, aber nicht 1. Eigentlich ist sie nicht wirklich zufällig, sondern nur *unecht* zufällig, und die Zahlen wiederholen sich nach jeweils 65537 Durchgängen, aber in der Praxis bemerken Sie das nicht. Da niemand weiß, wie eine "Zufalls"zahl wirklich aussieht, ist der Gebrauch von "unecht" selbst ein bißchen unecht.

Sie können das in Spielprogrammen verwenden. Beispiel: Um zu simulieren, daß Würfel gerollt werden, stellen Sie fest, daß 6 * RND eine Zufallszahl zwischen Ø und 6 ist (6 nicht eingeschlossen); demnach ist INT (6 * RND) zufällig Ø oder 1, 2, 3, 4, 5; demnach ist 1 + INT (6 * RND) zufällig eine Zahl von 1, 2, 3, 4, 5, 6. Genau das tut ein Würfel. Um eine Zufallskarte aus einem Paket von 52 Karten zu ziehen, würden Sie ein ähnliches Spiel mit 52 * RND spielen, aber Sie brauchen ein raffiniertes Programm, um die Zahlen zwischen Ø und 51 in Kartennamen wie "Pikbube" umzuwandeln. Es geht aber, wenn Sie geschickt sind.

Sie können Zufallszahlen auch für statistische Simulationen verwenden. Ein hübsches Beispiel dafür steht auf Seite 148, MONOPOLY-WÜRFELN.

9 VERZWEIGUNG

Manchmal hängt das, was Sie vom Computer verlangen wollen, von den Dingen ab, die bis dahin geschehen sind. In diesem Fall bedienen Sie sich einer Methode mit dem Namen Verzweigung.

Dieser Abschnitt befaßt sich mit Logik und *bedingten* Befehlen, wo das, was der Computer zu tun hat, von verschiedenen anderen Dingen abhängt. Beispiel Kinobesuch: IF (wenn) Sie unter 16 Jahre alt sind, THEN (dann) wird man Sie nicht in einen nur für Erwachsene freigegebenen Film hineinlassen. Ähnliche Bedingungen können Sie dem auferlegen, was der Computer macht.

Der Befehl dafür lautet IF . . . THEN . . . Auf das, was anstelle der Punkte steht, kommt es an.

Wir wollen ein Programm schreiben, das uns sagen soll, ob eine gegebene Zahl gerade oder ungerade ist. ("Auch schon was!" werden Sie sagen, und das stimmt ja auch, aber wie anderswo in diesem Buch geht es ums Prinzip, nicht um das konkrete Ergebnis.)

Dann mal los.

```
1Ø  INPUT n
2Ø  IF n = 2 * INT (n/2) THEN PRINT "gerade"
3Ø  IF n < > 2 * INT (n/2) THEN PRINT "ungerade"
```

Wie funktioniert das? Gerade Zahlen sind solche, die man ohne Rest durch 2 teilen kann. Demnach ist n also dann gerade, wenn n/2 eine ganze Zahl ist. Das heißt, mit INT bleibt sie gleich, also n/2 = INT (n/2). Und *das* ist dasselbe wie n = 2 * INT (n/2). Zeile 2Ø ist also nur eine kompliziertere Fassung von

```
2Ø  IF n gerade THEN PRINT "gerade"
```

was für uns ja durchaus Sinn ergibt – aber eben nicht für den armen alten Spectrum, der keine Ahnung hat, was "ist gerade" bedeutet, bis wir es ihm in der Sprache sagen, die er versteht.

Nur um das ganz deutlich zu machen, wollen wir ein paar Fälle berechnen.

n	22	23	24	25	26
n/2	11	11.5	12	12.5	13
INT n/2)	11	11	12	12	13
2 * INT (n/2)	22	22	24	24	26

Das sollte eigentlich genügen, um sogar den ärgsten Skeptiker zu überzeugen.

(Ebenso wird n durch k ohne Rest geteilt – bei ganzen Zahlen k, n – wenn und nur dann, wenn n = k * INT (n/k). Das zu wissen, ist sehr nützlich.)

Vielleicht sollte nun noch darauf hingewiesen werden, daß in Zeile 3Ø das Symbol < > "ist nicht gleich" bedeutet. Viele Mathematiker würden ≠ verwenden, aber der Spectrum bevorzugt < >. Fragen Sie mich nicht, warum.

Allgemein gesehen, tun Sie das Gleiche. Der entscheidende Befehl hat die Form

```
1Ø  IF dieses THEN jenes
```

wobei "dieses" und "jenes" Anweisungen sind. (In Zeile 2Ø oben ist "dieses" "n = 2 * INT (n/2)" und "jenes" "PRINT "gerade"".)

Die Anweisung "dieses" muß eine sein, die der Computer als entweder wahr oder falsch erkennen kann. (IF STOP THEN . . . sagt nicht sonderlich viel, im Gegensatz etwa zu IF n = 1981 THEN . . .). Wenn "dieses" wahr ist, geht der Computer weiter und tut "jenes"; ist "dieses" falsch, *geht er weiter zur nächsten Programmzeile, ohne "jenes" zu tun.*

Nun kommen zwei besonders häufige Arten von bedingten Befehlen.

Bedingte Sprünge

Sie haben die Form

```
1Ø  IF dieses THEN GO TO n
```

wobei n eine Zeilennummer ist. Solche Befehle kann man dazu verwenden, mitten im Strom die Pferde zu wechseln – das heißt, den ganzen Verlauf der Berechnung zu verändern, indem man zu einem anderen Teil des Programms geht.

Beispiel: Wenn man die Quadratwurzel einer Zahl berechnet, muß man berücksichtigen, daß negative Zahlen keine Quadratwurzeln haben. Sie vermeiden also Fehlermeldungen, wenn Sie so vorgehen:

```
1Ø  INPUT n
2Ø  IF n < THEN GO TO 5Ø
3Ø  PRINT n, SQR n
4Ø  STOP
5Ø  PRINT "Quadratwurzel nicht definiert"
```

Beachten Sie das STOP in Zeile 4Ø. Warum ist es vorhanden? Nehmen Sie es heraus und sehen Sie sich an, was geschieht!

Ebenso können Sie sich, wenn Sie PRINT AT a, b verwenden, dagegen schützen, daß a und b in nicht anzeigbaren Bereichen sind, wenn Sie schreiben:

```
1ØØ  IF a < Ø THEN GO TO 1ØØØ
11Ø  IF a > 21 THEN GO TO 1ØØØ
12Ø  IF b < Ø THEN GO TO 1ØØØ
13Ø  IF b > 31 THEN GO TO 1ØØØ
15Ø  PRINT AT a, b: "*"
```

```
 160  STOP
1000  das, was Sie als nützliche Antwort richtig finden . . .
      etc.
```

Wir gehen hier davon aus, daß a, b schon in einem vorangegangenen Programmteil zugeteilt wurden.

Setzen Sie davor

```
10  INPUT a
20  INPUT b
```

und befehlen Sie dem Computer dann PRINT AT 999, – 37, indem Sie a = 999, b = –37 setzen.

Geben Sie Zeile 1000 den Text

```
1000  "So dumm bin ich nicht mehr!"
```

Ist er's?

Wenn Sie finden, daß die Zeilen 100–130 plump aussehen, haben Sie ganz recht. Vergleiche den Abschnitt unten über LOGIK.

Bedingte Zuweisungen

Sie haben die Form IF dieses THEN LET irgend etwas. Beispiel: Eine Alternative zum ersten Programm oben wäre

```
10  INPUT n
20  LET a$ = "ungerade"
30  IF n = 2 * INT (n/2) THEN LET a$ = "gerade"
40  PRINT a$
```

Hier hat a$ ein Dollarzeichen, weil es keine Zahl ist, sondern eine Zeichenkette, ein *String* (siehe Kapitel 17).

Was bedeutet dieses Programm?

```
10  LET s = INT (2 * RND)
20  IF s = 0 THEN LET a$ = "KOPF"
30  IF s = 1 THEN LET a$ = "WAPPEN"
40  PRINT a$
```

Sobald Sie ein Programm schreiben und das, was Sie tun wollen, davon abhängt, daß bestimmte Dinge geschehen oder nicht geschehen, denken Sie gleich an IF . . . THEN . . .

Logik

Ein großes Thema, in gelehrten Werken immer wieder untersucht . . . aber das Meiste davon brauchen wir nicht zu wissen. Der Spectrum beherrscht Logik – vermutlich besser als Sie oder ich. Vor allem kann er Anweisungen kombinieren, die in der "dieses"-Position eines IF dieses THEN jenes vorkommen, und zwar dadurch, daß er AND, OR oder NOT (also "und", "oder", "nicht" verwendet.

Grundregeln: p AND q ist nur wahr, wenn p wahr ist und q wahr ist.
p OR q ist wahr, wenn p wahr ist oder q wahr ist oder beide wahr sind.
NOT p ist nur wahr, wenn p falsch ist.

Der Spectrum behandelt NOT vor AND und AND vor OR. Er behandelt nahezu alles andere, was Ihnen einfällt, bevor er sich mit einer von diesen Anweisungen befaßt. Die Folge: Sie brauchen oft keine Klammern, um den Befehl klarzumachen.

Beispiel: Wir können die Zeilen 1ØØ–13Ø oben verbessern, indem wir sie in die Einzelzeile verwandeln

```
1ØØ  IF a < Ø OR a > 21 OR b < Ø OR b > 31 THEN GO TO 1ØØØ
```

Es gibt alle möglichen staunenswerten Möglichkeiten, die Logik des Spectrums zu nutzen, aber sie sind nur umständlich zu erklären, und da ich nicht viel Platz habe, höre ich hier (ungern) auf. Mit Kapitel 10 des Sinclair-Handbuchs können Sie einen Anfang machen, aber das ist bei weitem nicht die ganze Geschichte.

Mehrfachbefehl-Zeichen

Der Spectrum läßt zu, daß Sie in einer einzelnen Zeile mehr als einen Befehl unterbringen. Die Befehle müssen lediglich durch Doppelpunkte ":" getrennt werden. Das Programm in Kapitel 1 über Basis-BASIC *hätte* man also auch etwa so schreiben können

```
1Ø  PRINT "Verdoppeln": INPUT x: LET y = x + x
2Ø  PRINT x, y: STOP
```

Wenn man wollte, könnte man auch das alles in eine einzige Zeile setzen.

Man kann das nutzen, um ein wenig Platz zu sparen (damit fallen Zeilennummern weg) oder um ein Programm leichter verständlich zu machen. Der größte Haken dabei ist, daß Sie mit GO TO nur zum *Beginn* einer Mehrfachbefehl-Zeile gehen können. Im wesentlichen habe ich Mehrfachbefehle vermieden – in der Regel kann man ohne sie leichter erkennen, was vorgeht – aber bei einer Gelegenheit können sie wirklich sehr nützlich sein. Ich habe diesen Kniff schon in Debugging 1 verwendet:

```
34  IF d = Ø THEN PRINT "Geht nicht": GO TO 30
```

Damit kann der Computer zwei verschiedene Dinge tun, wenn eine gegebene Bedingung gilt; die Verwendung vieler GO TOs wird vermieden.

Der wesentliche Punkt, den man sich merken muß: Nach einem THEN sind alle Befehle in dieser Zeile durch die IF-Anweisung bedingt, die dem THEN vorangeht. Mit anderen Worten: Ein Befehl

IF dieses THEN jenes: das nächste: etwas anderes

führt dazu, daß *alle drei*, also jenes, das nächste und etwas anderes ausgeführt werden (wenn dieses wahr ist) oder *überhaupt nichts* (wenn dieses falsch ist).

Nützlich, aber auch ein Stolperstein. Es fällt sehr leicht, Programme wie dieses zu schreiben:

```
1Ø   IF x = Ø THEN GO TO 2Ø: IF x = 1 THEN GO TO 1ØØØ
2Ø   PRINT "x ist Null"
     etc.
```

unter dem Eindruck, der Computer werde eine Verzweigung zu 2Ø machen, wenn x = Ø, und zu 1ØØØ, wenn x = 1. Tut er nicht. Wenn x – 1, dann faßt er Zeile 1Ø in diesem Sinn auf:

IF x = Ø THEN . . .
. . . GO TO 2Ø *und* IF x = 1 THEN GO TO 1ØØØ

aber

IF x < > Ø (was der Fall ist, wenn x = 1) THEN . . .
. . . beachte den Rest der Zeile nicht und geh zur nächsten.

Das ist hier Zeile 2Ø, genau jene, wohin er nicht gehen sollte!

Wenn Sie 1Ø aber ersetzen durch

```
1Ø   IF x = Ø THEN GO TO 2Ø
15   IF x = 1 THEN GO TO 1ØØØ
```

läuft alles so, wie Sie es erwarten.

Moral: *Zeilen mit Mehrfachbefehlen sind am nützlichsten in IF/THEN-Befehlen, dort aber auch am gefährlichsten.*

10 GRAFISCH DARSTELLEN

Einer der Hauptgründe, sich einen Spectrum zu kaufen: Seine ausgezeichnete Grafik. Fangen Sie an mit: Grafisch darstellen.

Für einen Kleincomputer ist der Spectrum mit ein paar recht verfeinerten Werkzeugen für das Zeichnen von Bildern ausgestattet. Er beherrscht Grafik mit sogenanntem hohem Auflösungsvermögen und auch mit geringem. Das mögen brauchbare Ausdrücke für Unterhaltungen bei Cocktailparties sein, aber in Wirklichkeit bedeutet es, daß man Bilder mit Bleistift zeichnen kann (hochauflösend) oder mit breitem Pinsel (geringe Auflösung).

In diesem Kapitel befasse ich mich nur mit der hochauflösenden Art von Grafik. Wenn Sie in diesen Modus umschalten, wird der Bildschirm aufgeteilt in eine große Zahl kleiner Quadrate, die Pixel heißen, 256 horizontal und 176 vertikal. Es gibt insgesamt also 256 x 176 = 45056 Pixel (Pixel von picture elements = Bildelemente).

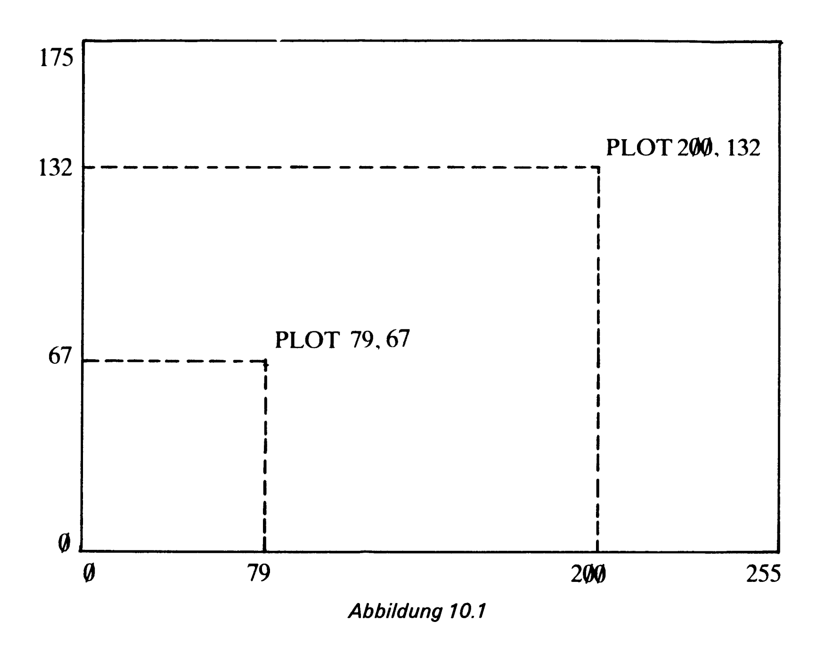


Abbildung 10.1

Abbildung 10.1 zeigt, wie auf sie Bezug genommen wird. Die erste Spalte wird mit Ø bezeichnet, die letzte mit 255. Die unterste Reihe hat die Bezeichnung Ø, die oberste 175. Auf dem Bildschirm wird ein Punkt dadurch geschwärzt, daß man beispielsweise schreibt:

```
5Ø  PLOT 79, 67
```

Wie Sie aus dem Schaubild erkennen können, zeigt der erste Wert nach PLOT die Spaltennummer an, der zweite Wert die Reihennummer. Das Schlüsselwort PLOT teilt dem Computer mit, daß Sie von ihm verlangen, er solle in Begriffen hoher Auflösung denken; für geringe Auflösung können Sie PLOT nicht verwenden. Es gibt für hohe Auflösung noch zwei andere Schlüsselwörter: DRAW und CIRCLE.

Befassen wir uns zuerst mit DRAW. In der einfachsten Form wird DRAW dazu benützt, eine gerade Linie zu zeichnen. Der Anfangspunkt für die Linie ist gegeben; er befindet sich immer dort, wo der "Bleistift" im Augenblick gerade ist. Die beiden Werte nach DRAW geben die Zahl der Spalten und Zahl der Reihen an, die zurückzulegen sind, bis das Ende der Linie erreicht ist. Demnach würde

```
10  PLOT 30, 30
20  DRAW 40, 80
```

eine Linie hervorbringen, wie Abbildung 10.2 sie zeigt.

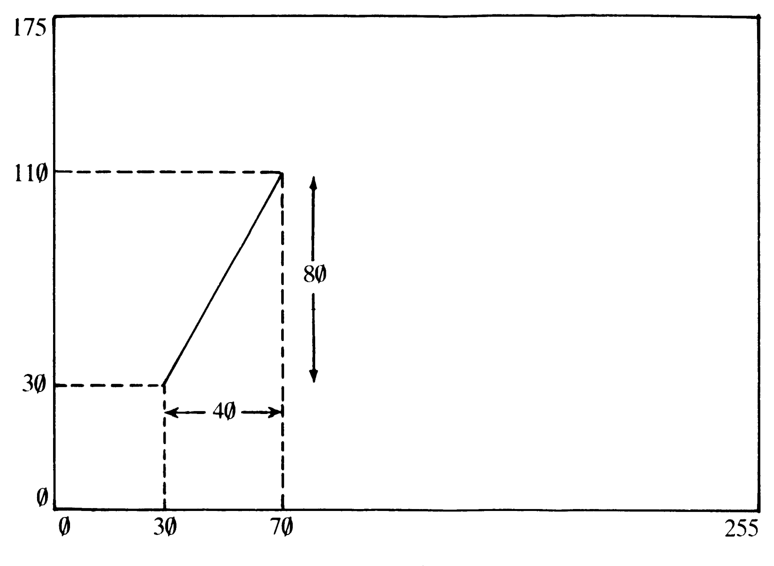


Abbildung 10.2

Fügen wir hinzu

```
30  DRAW 50, -10
```

so sind wir bei Abbildung 10.3.

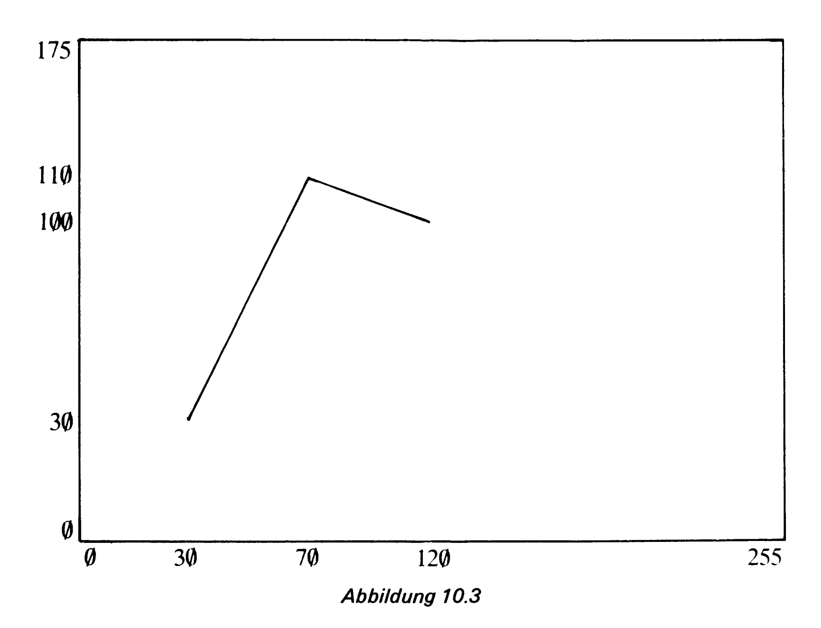


Abbildung 10.3

Es ist also ganz leicht, nach Wunsch etwa Rechtecke zu zeichnen. Angenommen, wir geben zuerst die Einzelheiten des Rechtecks ein und nennen die Werte an seiner linken unteren Ecke, seine Breite und seine Höhe:

```
10  INPUT "Spalte links außen:"; ls
20  INPUT "Unterste Reihe:"; ur
30  INPUT "Hoehe:"; h
40  INPUT "Breite:"; b
```

Zuerst können wir die untere linke Ecke zeichnen:

```
50  PLOT ls, ur
```

Jetzt die Grundlinie des Rechtecks:

```
60  DRAW b, 0
```

die rechte Seite:

```
70  DRAW 0, h
```

die obere Linie:

```
80  DRAW –b, 0
```

und die linke Seite:

```
90  DRAW 0, –h
```

Wenn das Rechteck eingeschwärzt werden soll, wird die Sache ein bißchen kniffliger. Wir müssen jetzt alle vertikalen Linien zwischen linkem und rechtem Rand zeichnen. Nehmen wir an, wir wollen nur eine davon zeichnen, und zwar die in einer Spalte, die s genannt wird. (Alles, was wir im Augenblick über s wissen, ist, daß es größer ist als ls, die Spalte mit dem linken Rand darin, und kleiner als ls + b, wo sich der rechte Rand befindet.) Der Code würde lauten:

```
110  PLOT s, ur +1     (um den Bleistift an die richtige
                        Stelle zu setzen)
120  DRAW 0, h -1
```

Das müssen wir nun bei allen Werten für s von ls +1 bis ls +b -1 tun. +1 und -1 stehen da, weil es keinen Sinn hat, die Ränder noch einmal zu zeichnen. Offensichtlich können wir eine FOR-Schleife gebrauchen:

```
100  FOR s = ls+1 TO ls+b-1
130  NEXT s
```

Um den Prozeß zu wiederholen, damit wir Rechtecke auf dem ganzen Schirm zeichnen können, brauchen wir nur anzufügen:

```
140  GO TO 10
```

Selbstverständlich werden die Rechtecke immer ausgefüllt, weil die Zeilen 100 bis 130 stets ausgeführt werden. Wir könnten den User fragen, ob er sein Rechteck auf folgende Weise einschwärzen will:

```
48  INPUT "Schwaerzen? (ja/nein)"; b$
```

und den "Schwärz"-Code ignorieren, wenn die Antwort "Nein" lautet:

```
95  IF b$ = "nein" THEN GO TO 10
```

Bei alledem wird unterstellt, daß der User sich vernünftig verhält und nicht versucht, Linien außerhalb der Bildschirmgrenzen zu zeichnen. Das ist eine gefährliche Unterstellung. User sollte man nicht so sehr als Trottel betrachten, die etwas falsch machen könnten, sondern als böswillige Menschen, die ganz gewiß alles tun, was sie können, um Ihr Programm zusammenbrechen zu lassen, wenn man ihnen nur die kleinste Möglichkeit dazu gibt.

Wir sollten also ein paar Tests einschalten, um dafür zu sorgen, daß das angegebene Rechteck auch gezeichnet werden kann, bevor wir es damit versuchen.

Erstens: Ist es die Bildschirmspalte ganz links außen?

```
15  IF ls<0 OR ls>255 THEN PRINT "geht nicht": GO TO 10
```

Dann: Ist die unterste Reihe möglich?

```
25  IF ur<0 OR ur>175 THEN PRINT "geht nicht": GO TO 20
```

Ist die Höhe negativ?

```
33  IF h<0 THEN PRINT "ist doch albern!": GO TO 30
```

Paßt die oberste Zeile hinein?

```
36  IF ur+ h>175 THEN PRINT "paßt nicht rein": GO TO 30
```

Paßt der rechte Rand?

```
44  IF ls+b>255 THEN PRINT "nicht zu machen": GO TO 40
```

Außerdem sollten wir uns vergewissern, daß als Antwort auf die Frage in Zeile 48 nur "ja" oder "nein" eingegeben worden sind:

```
49  IF b$< > "ja" AND b$ < > "nein" THEN PRINT
    "Bitte ja oder nein eingeben": GO TO 48
```

Hier ist ein interessanter Punkt, auf den Sie achten sollten. Für den Computer ist "ja" keineswegs dasselbe wie "JA". Wenn wir das Programm also so lassen, wie es ist, und der User CAPS LOCK geschaltet hat (d. h. alles wird in Großbuchstaben geschrieben), tippt er als Antwort auf Zeile 48 "JA" oder "NEIN", und die ärgerliche Maschine erwidert jedesmal:

```
Bitte ja oder nein eingeben
Schwaerzen? (ja/nein)
```

Probieren Sie, ob Sie Zeile 49 so abändern können, daß der User entweder große oder kleine Buchstaben eingeben kann.

Kreisen um den Spectrum

Kreise zu zeichnen, ist sehr einfach. Es gibt ein besonderes Schlüsselwort CIRCLE, nach welchem Sie mit Spalte und Reihe angeben, wo der Mittelpunkt sein soll, und dann den Radius. Zum Beispiel zeichnet

```
20  CIRCLE 50, 70, 30
```

einen Kreis, dessen Mitte am Schnittpunkt von Spalte 50 und Reihe 70 liegt, und der den Radius 30 hat.

DRAW kann aber auch dazu verwendet werden, Kreise oder vielmehr Teile von Kreisen zu zeichnen. Versuchen Sie einmal:

```
10  PLOT 20, 30
20  DRAW 60, 100
30  DRAW 60, 100, 1
```

Sie werden sehen, daß Zeile 2Ø wie erwartet eine gerade Linie zeichnet und Zeile 3Ø zu dem ursprünglich mit PLOT festgelegten Punkt zurückkehrt, was ebenfalls zu erwarten war. Das geschieht aber auf einer Kreisbahn und nicht entlang einer geraden Linie. Was BASIC anweist, einen Kreisbogen zu beschreiben, ist die dritte Variable nach DRAW. Wie das geht, ist ein bißchen verwirrend, aber ich will es trotzdem erklären.

Stellen Sie sich den ganzen Kreis vor (den Teil, der nicht gezeichnet wird, zeigt Abbildung 10.4 gestrichelt).

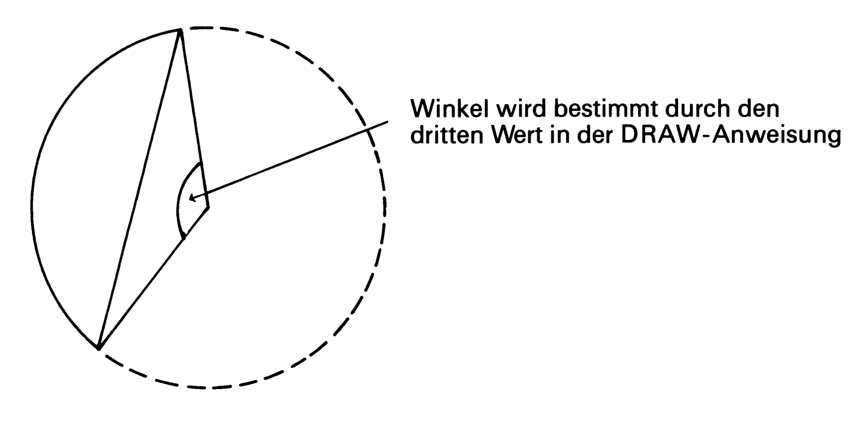


Abbildung 10.4

Wenn Sie Linien vom Mittelpunkt bis zu den Enden des Bogens ziehen, ist der Winkel zwischen ihnen der dritte Wert nach DRAW. Das mag Sie überraschen, weil dieser Winkel ganz offensichtlich viel größer ist als 1. Das liegt daran, daß der Winkel (damit das Leben nicht zu einfach wird), in Radianten gemessen wird, und 1 Radiant knapp unter 6Ø° beträgt. Falls das nun alles nicht kristallklar sein sollte und Sie sich unter Radianten etwas ganz anderes vorgestellt haben, zerbrechen Sie sich deswegen nicht den Kopf. In der Praxis bedeutet das nur, daß Sie dann, wenn Sie einen kleinen Winkel wählen (meinetwegen Ø.1), eine Linie erhalten, die von einer geraden nur wenig abweicht. Je größer der Winkel, desto deutlicher die Kreisform der Linie. Probieren Sie:

```
1Ø  PLOT 6Ø, 2Ø
15  FOR a = Ø.4 TO PI STEP Ø.4
2Ø  DRAW 1ØØ,Ø
3Ø  DRAW –1ØØ,Ø, a
4Ø  NEXT a
```

Sehen Sie, wie die Form sich einem Halbkreis annähert, wenn der Winkel an PI (3.14159) herankommt?

Ändern Sie Zeile 15 jetzt ab zu:

```
15  FOR a = Ø.4 TO 2*PI STEP Ø.4
```

Sie erhalten immer mehr einen Vollkreis, bis er am Ende nicht mehr auf den Schirm paßt. Es spielt keine Rolle, wie klein Sie die gerade Linie zunächst wählen, früher oder später paßt der Kreis nicht mehr auf den Schirm. Der Grund dafür: 2*PI Radianten sind 36Ø° Grad – ein voller Kreis! Da dieser volle Kreis die anfängliche *gerade* Linie enthält, muß sein Radius unendlich sein. Er paßt also überhaupt nirgends hinein! Die Moral: Lassen Sie den Winkel nicht zu groß werden (um 5 liefert den größten Teil eines Kreises), und seien Sie selbst dann vorsichtig; man kann leicht vom Bildschirm rutschen. Wohin man dann gerät, ist nicht leicht festzustellen.

Löcher ausfüllen

Wir haben gesehen, wie man ein Rechteck schwärzt, aber einen Kreis oder ein Kreissegment zu schwärzen, scheint doch eine erheblich schwierigere Aufgabe zu sein. Der Grund: Man kann nicht so leicht erkennen, wo der DRAW-Befehl für das Schwärzen anfangen und aufhören muß, weil diese Werte nicht so klar festgelegt sind wie bei einem Rechteck. Und das ist der Schlüssel zur Lösung. Wir müssen herausbekommen, wo die in jeder Reihe die Ränder der Figur sind, die wir schwärzen wollen. (Zur Abwechslung schwärzen wir einmal nicht die Spalten, sondern die Reihen.)

Zum Glück gibt uns der Spectrum eine Möglichkeit für die Feststellung, ob ein bestimmtes Pixel geschwärzt wird. Das ist die POINT-Funktion. So setzt

```
2ØØ   LET G = POINT (2Ø, 3Ø)
```

g auf 1, wenn das Pixel bei 2Ø, 3Ø geschwärzt wird, und auf Null, wenn nicht.

Das Problem gliedert sich also so auf:

1 Legen Sie einen rechteckigen Raum um die zu schwärzende Figur fest, in dem die Suche nach den Rändern ablaufen soll.
2 Tun Sie bei jeder Reihe Folgendes:
   a) Suchen Sie von links her, bis die Figur getroffen wird. Stellen Sie fest, wo das ist.
   b) Suchen Sie von rechts her, bis die Figur getroffen wird. Stellen Sie fest, wo das ist.
   c) Ziehen Sie eine Linie zwischen den beiden nach a) und b) gefundenen Punkten.

Hier das Programm, das dabei herauskommt:

```
6ØØ   INPUT "Spalten rahmen"; ls, rs        [linke und rechte Spalten
                                            des Rahmenrechtecks]
61Ø   INPUT "Reihen rahmen"; ur, or         [unterste und oberste Rei-
                                            hen des Rahmenrechtecks]
62Ø   FOR r = ur TO or
63Ø   FOR s = ls TO rs                      [Suche von links nach
64Ø   IF POINT (s, r) = 1 THEN GO TO 66Ø    rechts nach dem Rand]
65Ø   NEXT s
```

655 GO TO 730	[Wenn das Programm hier ankommt, gibt es in dieser Reihe keinen Rand, also zur nächsten gehen]
66Ø LET s1 = s	[s1 ist äußerste Spalte links]
67Ø FOR s = rs TO ls STEP –1	[Suche von rechts nach
68Ø IF POINT (s, r) = 1 THEN GO TO 7ØØ	links nach dem Rand]
69Ø NEXT s	
7ØØ LET s2 = s	[s2 ist äußerste Spalte rechts]
71Ø PLOT s1, r	[ein-
72Ø DRAW s2–s1,Ø	schwärzen]
73Ø NEXT r	

Natürlich müssen Sie eine Routine zum Zeichen einer geschlossenen Figur voransetzen, vielleicht die für das Zeichen eines Kreissegments zu Beginn des vorigen Abschnitts, damit überhaupt etwas stattfindet.

Man kann einige leichte Veränderungen vornehmen, mit denen allerhand herauszuholen ist.

Ändern Sie als erstes Zeile 62Ø ab:

```
62Ø   FOR r = ur TO or STEP c
```

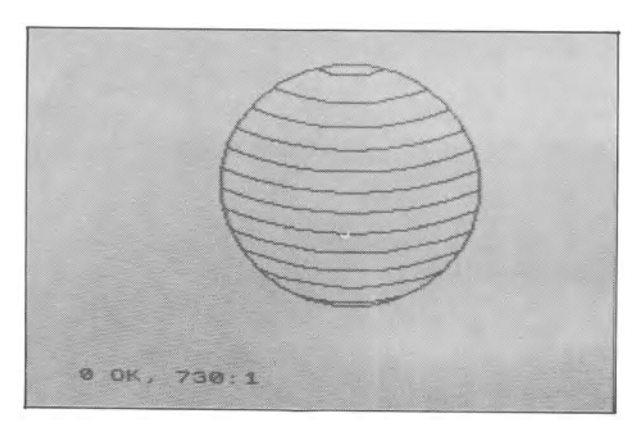
und schreiben Sie dann eine Zeile, die es dem User gestattet, jeden beliebigen Wert von s einzugeben (an einer passenden Stelle vor 620). Wird c auf 1 gesetzt, bleibt die Wirkung gleich, aber bei c = 2 wird nur jede zweite Reihe eingezeichnet, so daß wir statt der Schwärzung eine Schraffierung erhalten. Bei größeren Werten von c werden die Schraffierungszwischenräume natürlich breiter.

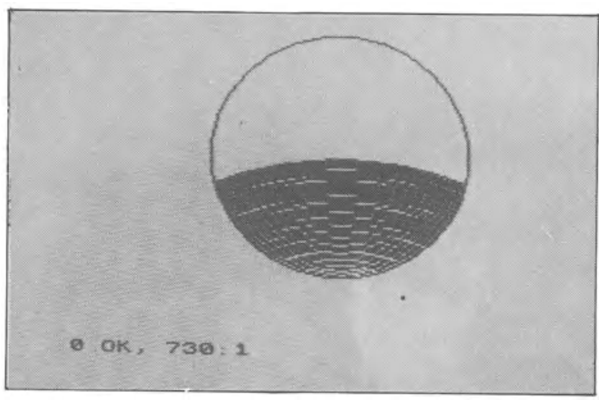
Nun schreiben Sie Zeile 72Ø um:

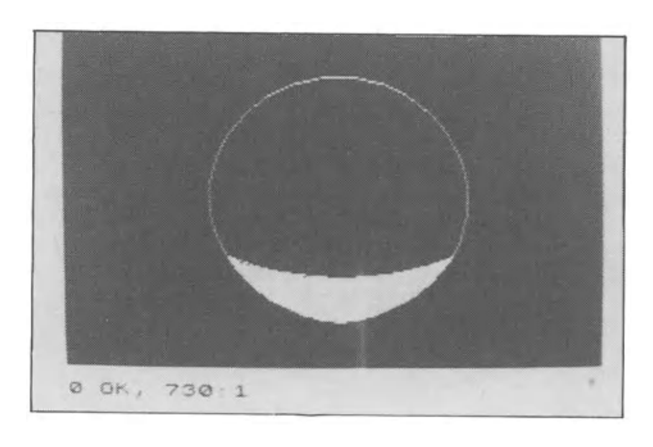
```
72Ø   DRAW s2 s1,Ø, a
```

und fügen Sie für "a" an passender Stelle eine INPUT-Zeile ein. Versuchen Sie nun einen Kreis mit einem ziemlich kleinen Wert von "a" (sagen wir Ø.5) zu schraffieren. Sehen Sie die dreidimensionale Wirkung?

Wählen Sie nun das Rahmenrechteck so, daß die obere Kreishälte darüberliegt. Verwenden Sie denselben Wert von a, setzen Sie c aber auf 1 zurück. Jetzt haben Sie einen Mond im Schatten. (Verwenden Sie Ink weiß auf Paper schwarz, damit die Wirkung besonders gut herauskommt.) Und so weiter und so fort. Als ich dieses Programm zum erstenmal schrieb, hatte ich viel Spaß dabei, Gänsekiele in Tintenfässern zu zeichnen. Das Schlichte eben . . .


0 OK, 730:1


0 OK, 730:1


0 OK, 730:1

Der elektronische Radiergummi von BASIC

Zuzusehen, wie Formen gelöscht werden, macht fast soviel Spaß wie das Zeichnen. Sie verwenden auch dafür die Befehle PLOT, DRAW und CIRCLE, müssen BASIC aber sagen, daß Sie jetzt nicht den Bleistift in der Hand haben, sondern den Radiergummi.

Probieren Sie Folgendes aus:

```
50  CIRCLE 120, 85, 60
60  OVER 1
70  CIRCLE 120, 85, 60
```

Sie werden sehen, wie der Kreis zuerst gezogen und dann ausradiert wird. Ersetzen Sie Zeile 70 jetzt durch GO TO 50 und lassen Sie wieder laufen. Der Kreis wird gezogen, ausradiert, neu gezogen und so weiter!

Na gut, ich habe geschwindelt. In Wirklichkeit radiert der Computer im Modus "OVER 1" aus, wenn etwas zum Radieren da ist, sonst zeichnet er. Sobald Zeile 50 also das erstemal zum Tragen kommt, wird der Kreis gezeichnet, beim nächstenmal ausradiert (weil er jetzt vorhanden ist), beim nächstenmal gezogen (weil es nichts auszuradieren gibt), und so weiter.

Jetzt nehmen Sie:

```
10  OVER 0                    [um das Radieren abzustellen]
20  CIRCLE 40, 40, 30
30  CIRCLE 60, 40, 30
40  OVER 1                    [Radierer einschalten]
50  CIRCLE 100, 100, 30
60  CIRCLE 120, 100, 30
```

Sehen Sie die Wirkung? Wenn der Radierer abgeschaltet ist, überlappen sich die beiden ersten Kreise der Erwartung entsprechend. Ist der Radierer eingeschaltet, überlappen sich auch die beiden anderen Kreise, aber der zweite radiert Teile des ersten dort aus, wo sie sich schneiden.

Die Wirkung von CIRCLE (oder DRAW oder PLOT) bei OVER ist demnach: "Wenn etwas zu löschen da ist, dann lösche es; sonst zeichne die gewünschte Form." Im obigen Programm hält die Wirkung des "OVER 0" in Zeile 10 an, bis es durch das "OVER 1" in Zeile 40 widerrufen wird. Man kann einen "OVER"-Befehl für nur eine Anweisung gelten lassen, und zwar so:

```
70  CIRCLE OVER 0; 130, 100, 20
```

Der neue Kreis überlappt den vorherigen, ohne die Schnittstellen zu löschen, aber bei jeder folgenden Anweisung, die keine OVER-Bestimmung enthält (probieren Sie beispielsweise 90 PLOT 0, 0: DRAW 150, 148) bleibt das OVER 1 in Zeile 40 in Kraft und radiert somit an Schnittstellen, auch wenn man dort, wo die Schnittstelle ein einziges Pixel ist, schon ganz genau hingucken muß, um die Wirkung zu erkennen. Wie sähe die Wirkung aus, wenn man das "OVER 0;" aus Zeile 70 herausnähme? Versuchen Sie es einmal!

Wie man seine Kunstwerke für die Nachwelt aufbewahrt

Sie können die Techniken, die wir besprochen haben, dazu benützen, recht phantasievolle Muster zu zeichnen. In "Fertigprogramme" finden Sie ein Programm, mit dem das ziemlich leicht geht (es nutzt die Prozeduren für Schwärzen und Schraffieren, die hier beschrieben worden sind).

Es wäre also praktisch, wenn wir die Resultate auf Band speichern könnten. Beispielsweise könnten wir ein wunderschönes Mondlandschafts-Display entwerfen, das wir vielleicht bei einem Mondlande-Programm verwenden wollen, oder wir möchten eine Reihe von Bildern haben, um mit jedem ein Loch auf einem Golfplatz darzustellen. Ein Golfprogramm könnte sie der Reihe nach so laden, wie sie gespielt werden.

Der Spectrum erleichtert dieses Verfahren sehr. Wir sichern und laden den Inhalt des Displayschirms fast genauso, wie wir Programme sichern und laden. Der einzige Unterschied ist der, daß wir nach den normalen SAVE- und LOAD-Befehlen das Wort "SCREEN$" schreiben, um anzuzeigen, daß es der Bildschirm ist, den wir auf Band genommen (oder zurückgelesen) haben wollen und kein Programm. Zum Beispiel sagt:

```
SAVE "GLOCH3" SCREEN$
```

"sichere den Bildschirminhalt als eine Banddatei namens "GLOCH3", und, wenn man zurückladen will:

```
LOAD "GLOCH3" SCREEN$
```

11 DEBUGGING II

Bleistift und Papier sind nach wie vor von Nutzen . . .

Vorhin, auf Seite 35, habe ich Sie mit dem Mittelwertprogramm allein gelassen. Es ist unten noch einmal abgedruckt, damit Sie nicht zuviel blättern müssen. Ich hatte den Vorschlag gemacht, Sie könnten es einmal mit dem Debugging versuchen.

```
 1Ø  LET s = Ø
 2Ø  LET c = Ø
 3Ø  INPUT n
 4Ø  IF n<Ø THEN GO TO 1ØØ
 5Ø  LET c = c+i
 6Ø  LET z = s+n
 7Ø  GO TO 3Ø
1ØØ  PRINT "MITTEL IST □"; s/c
```

Überlegen wir uns ein schrittweises Vorgehen bei der Lösung des Problems. Offenkundig kommt es als erstes darauf an, festzustellen, ob es so funktioniert, wie es dasteht. Wir geben es also in den Spectrum ein und fahren es mit ein paar einfachen Datenmengen. Versuchen wir es mit:

3

7

5

–1 [beachten – dieser Wert dient nur als Terminator oder Begrenzer]

Darauf *sollte* die Antwort 5 sein. Wenn das Programm gefahren wird, erhalten Sie jedoch die Fehlermeldung 2 Variable not found, 5Ø:1 (also "Variable nicht gefunden"). Hmm. Das heißt: In Zeile 5Ø ist eine bis dahin nicht definierte Variable verwendet worden. Aus dem Listing können wir ersehen, daß dort i steht. Geben wir i also den Wert, meinetwegen.

```
5  LET i = Ø
```

Das sorgt dafür, daß das Programm über Zeile 5Ø hinaus fortfährt. Wenn wir Zeile 5 anfügen und wieder RUN drücken, stellen wir fest, daß die Meldung

MITTEL IST

zwar richtig angezeigt wird, dann aber eine Fehlermeldung folgt, nämlich

6 Number too big, 1ØØ:1

Diese Meldung kennen wir schon. Sie werden sich vermutlich erinnern: Der "6" zufolge hat der Computer versucht, eine Berechnung anzustellen, deren Resultat er nicht aufnehmen kann. "1ØØ" gibt die Zeilennummer an, wo das Problem aufgetreten ist. Man wird also vermuten dürfen, daß das Programm versucht hat, durch Null zu teilen, wie schon beim letztenmal, als diese Fehlermeldung aufgetaucht ist. Bevor wir uns als echte Detektive betätigen, wollen wir noch mehr Beweismaterial sammeln. Versuchen Sie es mit anderen Daten:

2

1

4

6

–1

Die Meldungen sehen genauso aus wie vorher. Als Arbeitshypothese scheint sich demnach anzubieten: "Gleich, welche Daten eingegeben werden, das Problem schließt mit einer Meldung über arithmetischen Überlauf ab." Testen Sie das Programm mit drei oder vier anderen Datenmengen. Unsere Arbeitshypothese scheint sich immer mehr zu festigen.

Wo fangen wir also mit dem Suchen an? Vorhin habe ich die Ausdrücke "Detektiv" und "Beweismaterial" verwendet, und das nicht leichthin. Fehler in Programmen auszumerzen, gleicht der kriminalistischen Arbeit, und jeder Kriminalist wird Ihnen sagen, daß man wie ein Halunke denken muß, um Erfolg zu haben, oder wenn Sie so wollen: "Nur ein Dieb fängt einen Dieb." Der Halunke in diesem Stück ist der Spectrum, also geht es für uns darum, so zu denken wie er. Als erstes müssen Sie Ihre Denkprozesse verlangsamen. Das ist eine Überraschung für Sie, nicht wahr? Sie standen unter dem Eindruck, Computer wären eher schnell. Sind sie auch, aber die Art, wie sie über ein Problem "nachdenken", ist im Regelfall recht umständlich. Stellen wir als nächstes ein Modell des Computerspeichers oder wenigstens des Teils auf, der für das vorliegende Problem von Belang ist. In unserem Beispiel sind fünf Speicherelemente vorgesehen. Sie haben die Namen s, c, n, i und z erhalten. Ein brauchbares Modell wird also schlicht eine Tabelle sein, mit der wir zeigen können, wie die Inhalte dieser Elemente sich während der Ausführung des Programms verändern. Es kann ferner nützlich sein, einen Hinweis auf die Zeilennummer zu haben, wo eine Verzweigung stattfindet, wenngleich das nicht unbedingt erforderlich ist. Die Tabelle könnte also folgendermaßen aussehen:

Zeilennummer	s	c	n	i	z	Verzweigung

Ich habe eine zusätzliche Spalte "Verzweigung" angefügt, deren Funktion ich gleich erkläre. Wir stellen die Tabelle nun dadurch auf, daß wir uns der Reihe nach ansehen, was jede Anweisung bewirkt. Wir müssen eine Datenmenge definieren, mit der wir arbeiten können; nehmen wir:

2

1

4

6

–1

Die ersten Anweisungen, die ausgeführt werden müssen, stehen in den Zeilen 5 und 10 und setzen die Speicherelemente i und s auf Null. Die Tabelle sieht dann so aus:

Zeilennummer	s	c	n	i	z	Verzweigung
5				∅		
1∅	∅					

Nachdem Zeile 2∅ ausgeführt ist, haben wir:

Zeilennummer	s	c	n	i	z	Verzweigung
5				∅		
1∅	∅					
2∅		∅				

Zeile 3∅ übernimmt den ersten Wert der Datenmenge und stellt ihn in n, demnach:

Zeilennummer	s	c	n	i	z	Verzweigung
5				∅		
1∅	∅					
2∅		∅				
3∅			2			

Zeile 4∅ ist eine "IF"-Anweisung und dient damit nur zur Prüfung, ob n negativ ist. Das ist nicht der Fall (zur Zeit hat es den Wert 2), also ist der Test falsch, was wir in der Spalte "Verzweigung" durch ein "x" anzeigen, so daß der

Sprung zu 100 nicht stattfindet. Wenn ein Test richtig ist und der Sprung stattfindet, zeigen wir das in der Spalte "Verzweigung" durch ein "√" an. Jetzt haben wir:

Zeilennummer	s	c	n	i	z	Verzweigung
5				0		
10	0					
20		0				
30			2			
40						×

Zeile 50 verlangt vom Computer, er soll den Inhalt von i holen, den Inhalt von c addieren und das Resultat nach c zurückstellen. Das sollte uns ein bißchen argwöhnisch machen, weil wir wissen, daß i ursprünglich nicht festgesetzt worden ist. Zeile 5 anzufügen, war also vielleicht doch keine so gute Idee. Fahren wir aber mit unserem simulierten Programm trotzdem fort.

Zeilennummer	s	c	n	i	z	Verzweigung
5				0		
10	0					
20		0				
30			2			
40						×
50		0				

Null zu Null zu addieren, hat natürlich keine nutzbringende Wirkung, und das müßte uns noch argwöhnischer machen. Wir rufen uns ins Gedächtnis, daß wir nach Schreibfehlern suchen, und könnten auf den Gedanken kommen, i soll vielleicht gar nicht i sein, sondern 1, ein Versehen, das ja leicht vorkommen kann.

Wir fangen jetzt mit einer neuen Tabelle ganz von vorne an und gehen von der neuen Vermutung aus, Zeile 5 sei unnötig und Zeile 50 laute

```
50  LET c=c+1
```

Zeilennummer	s	c	n	z	Verzweigung
1∅	∅				
2∅		∅			
3∅			2		
4∅					×
5∅		1			
6∅				2	
7∅					√
3∅			1		
4∅					×
5∅		2			
6∅				1	
7∅					√
3∅			4		
4∅					×
5∅		3			
6∅				4	
7∅					√
3∅			6		
4∅					×
5∅		4			
6∅				6	
7∅					√
3∅			–1		
4∅					√
1∅∅					

Wenn das Programm Zeile 1∅∅ erreicht, zeigt es die Meldung "MITTEL IST" an, gefolgt vom Wert für s/c, der Null ist!

Das Programm funktioniert also immer noch nicht. Interessant dabei ist aber, daß der Spectrum keine Fehlermeldung mehr bringt; wir haben eine neue Art von Fehler eingeführt – einen Logikfehler. Der Spectrum kann die Operationen, die wir verlangt haben, zwar ausführen, aber er gelangt mit ihnen zur falschen Lösung.

Sehen wir uns die Tabelle an, die wir generiert haben. Jetzt ist ziemlich klar, was die Funktion von c ist. Jedesmal, wenn in n ein neuer Wert eingegeben wird, erhöht sich der Wert in c um einen Punkt. Sobald die Verzweigung zu Zeile 1∅∅ stattfindet, enthält c also die Zahl der insgesamt eingegebenen Werte, in diesem Fall 4. Mit s ist aber, seitdem es auf Null gestellt wurde, überhaupt nichts geschehen, und z enthält lediglich dieselben Werte wie n, aber ein bißchen später. Vielleicht sind s und z in Wahrheit identisch. Da s in Zeile 1∅ als erstes erwähnt wird, wollen wir unterstellen, z sei eine falsche Schreibweise für s. Demnach lautet Zeile 6∅:

```
6∅  LET s=s+n
```

und aus der Tabelle wird:

Zeilennummer	s	c	n	Verzweigung
1Ø	Ø			
2Ø		Ø		
3Ø			2	
4Ø				×
5Ø		1		
6Ø	2			
7Ø				√
3Ø			1	
4Ø				×
5Ø		2		
6Ø	3			
7Ø				√
3Ø			4	
4Ø				×
5Ø		3		
6Ø	7			
7Ø				√
3Ø			6	
4Ø				×
5Ø		4		
6Ø	13			
7Ø				√
3Ø			–1	
4Ø				√
1ØØ				

Angezeigt wird uns jetzt:

MITTEL IST 3.25

was zutrifft!

Den Vorgang, den ich beschrieben habe, nennt man "Schreibtischtest". Das ist eine oft angewandte Methode der Fehlersuche. Selbstverständlich muß man nicht immer alle Einzelheiten, die ich dargestellt habe, in die Tabelle eines Schreibtischtests aufnehmen (so sind etwa die Zeilennummern in diesem Fall für uns nicht sehr nützlich gewesen), und oft braucht man eine Tabelle nicht fertigzustellen, bevor es einem dämmert. Sie werden aber erkennen können, daß das ein hübscher Weg ist, Sie zum Zwangsjackendenken des Computers zu zwingen und gleichzeitig deutlich zu machen, wie das Programm abläuft.

Sie mögen nun sagen: "Alles schön und gut, aber wer macht denn solche Schreibfehler, wenn er Programme eingibt?"

Die Antwort: Das kommt dauernd vor, und zwar aus einer ganzen Reihe von Gründen. Erstens: Wenn Sie ein Programm aus einer Zeitschrift abschreiben, besteht die Möglichkeit, daß der Fehler schon ausgedruckt ist. Zweitens: In einem langen Programm kann Ihnen durchaus dieser oder jener Tippfehler unterlaufen – i für 1 (wenn das Listing, von dem Sie abschreiben, alle Großbuchstaben verwendet hat), den Buchstaben O für Ø, 2 für z und so weiter.

Drittens: Sogar dann, wenn Sie ein Programm selbst schreiben, können Sie einen Fehler machen, der auf einen Schreibfehler hinausläuft.

Beispiel: Nehmen Sie an, Sie nennen eine Variable zu Beginn eines Programms b3. Sie schreiben mehrere Tage an dem Programm, basteln hier herum, ändern dort eine Zeile ab. Am Ende müssen Sie noch ein paar Zeilen schreiben, die diese Variablen enthalten, und Sie wissen noch ganz genau, daß sie b2 hieß. Nachzusehen brauchen Sie da gar nicht mehr.

Das gibt es nicht, meinen Sie? Warten Sie mal ab, bis Sie ein paar Monate lang programmiert haben.

12 GRAFIK

Der Spectrum kann zeichnen. Sobald er weiß, was er zeichnen soll, kann er die Zeichnung auch bewegen. Sobald er sie bewegen kann, läßt die Bewegung sich mit dem Keyboard steuern. Vor allem dann, wenn Sie Computerspiel-Programme schreiben wollen, müssen Sie etwas von Grafik verstehen.

Ein besonders erfreulicher Zug von Computern ist der, daß sie Bilder zeichnen können – bei entsprechender Hardware sogar wunderschöne und komplizierte, vielfarbige Bilder. Der Spectrum hat auf diesem Gebiet zwar deutliche Grenzen, besitzt aber Fähigkeiten genug, um eine anregende Einführung in die Computergrafik zu bieten.

Kapitel 1Ø beschreibt einen der Wege, mit dem Computer zu zeichnen. Dort haben wir uns aber mit eher trockenen Dingen wie Rechtecken und Kreisen befaßt. Mit der PRINT-Anweisung lassen sich Zeichnungen machen, die dem Auge wohlgefälliger sind.

Der "PRINT"-Befehl

Der Grundbefehl hier ist PRINT x$, wobei x$ ein Zeichen oder ein Zeichenstring ist. Das erklärt sich eigentlich selbst. Wenn Sie mit PRINT experimentieren, werden Sie aber bald dahinterkommen, daß dieser Befehl allein Ihnen wenig Macht darüber verschafft, *wo* ein bestimmtes Zeichen angezeigt wird: Der Computer beginnt mit jedem Befehl solcher Art auf der linken Bildschirmseite in der nächstverfügbaren Zeile, also nicht immer dort, wo Sie ihn haben wollen.

Das besorgt PRINT AT x, y. Ganz ähnlich wie bei PLOT hat man sich den Bildschirm so vorzustellen, als sei er in Quadrate aufgeteilt, bezeichnet mit zwei Koordinaten x und y. Es gibt aber mehrere Unterschiede. Erstens einmal sind die Quadrate doppelt so groß. Das heißt: In jeder Richtung gibt es nur halb so viele, insgesamt also nur ein Viertel. Zweitens ist das Numerierungssystem für die Koordinaten ganz anderer Art – nämlich einfacher. Die erste Zahl x ist eine *Zeilen*nummer auf dem Schirm; sie reicht in dem Bereich, der für PRINT zur Verfügung steht, von Ø oben bis zu 21 unten. Die zweite Zahl, also y, ist eine *Spalten*nummer. Sie bezeichnet die Entfernung entlang einer Zeile (also horizontal) und reicht von Ø bis 31. Abbildung 12.1 zeigt dieses System ausführlich.

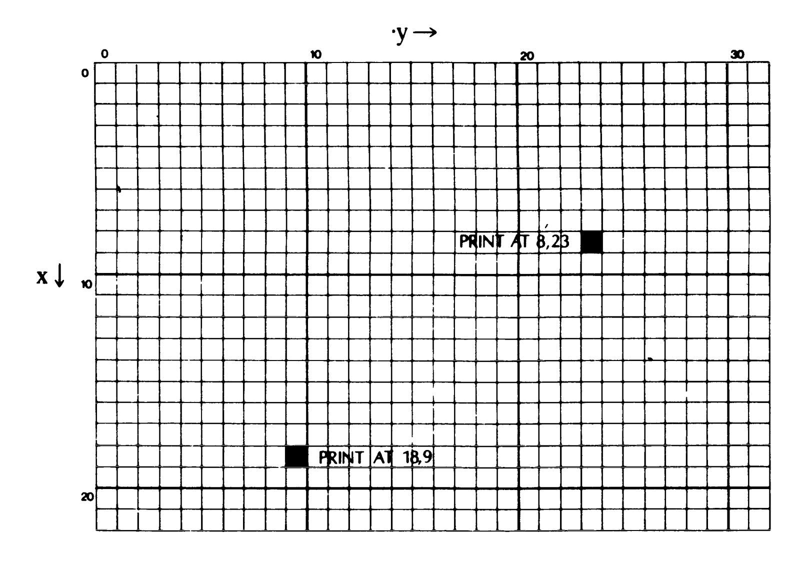


Abbildung 12.1

Nehmen wir an, Sie wollen das Grafikzeichen ■ mitten auf dem Bildschirm anzeigen. Die exakte Mitte ist nicht erreichbar, aber das Quadrat in Zeile 11 und Spalte 15 kommt ihr ziemlich nah. Sie würden also schreiben

```
10  PRINT AT 11, 15; "■"
```

Beachten Sie den Strichpunkt (;). Er ist erforderlich, um den Computer anzuweisen, gemeinsam das auszuführen, was er für *zwei* Befehle hält: geh nach 11, 15; zeige etwas an.

Wenn Sie mehrere Grafikzeichen aneinanderfügen, können Sie interessante Wirkungen erzielen. Der direkteste Weg dazu, nämlich die Verwendung vieler PRINT AT-Befehle, schluckt kostbaren Speicherplatz wie ein Cadillac Sprit, aber im Augenblick wollen wir uns den Kopf nicht über Wirtschaftlichkeit zerbrechen; wichtig ist nur das Prinzip. Probieren Sie dieses Programm aus:

```
10  PRINT AT 10, 10; "■■■▀▀▀▀"
20  PRINT AT 11, 8; "■■■■■■■"
30  PRINT AT 12, 8; "▐[0][0][0][0][0]▌"
```

Das müßte Sie an etwas Militärisches erinnern. (Falls nicht, dürfen Sie meinen schwachen Zeichenkünsten die Schuld geben.)

Wenn man solche Grafikbilder anzeigen will, gibt es dazu eine nützliche Vorbereitung: Zeichnen Sie das Bild auf kariertes Papier und numerieren Sie Zeilen und Spalten, dann sind die erforderlichen Befehle leichter abzulesen. Das obige Programm wird auf diese Weise von Abbildung 12.2 gewonnen.

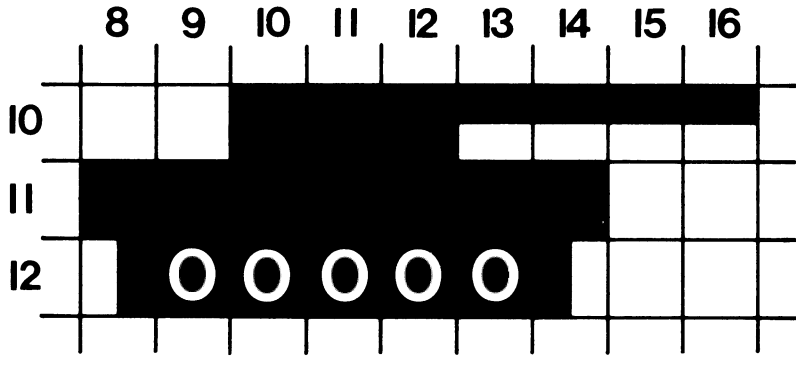


Abbildung 12.2

Der Spectrum hat 16 grafische Sonderzeichen (im Zeichenvorrat die Nummern 128-143 – siehe dazu Anhang A des Handbuchs oder Datenblatt). Sie können aber auch andere Zeichen gut verwenden (wie das obige Beispiel zeigt.) Video-Negativschrift (Weiß auf Schwarz) ist in diesem Zusammenhang besonders nützlich.

Die Position verändern

Wenn man einmal weiß, wie auf einer bestimmten Stelle auf dem Bildschirm mit PRINT etwas angezeigt werden kann, fällt es nicht mehr schwer, das Programm so abzuwandeln, daß die Grafikzeichen an jede gewünschte Stelle gesetzt werden können. Der obige Panzer (genau, das sollte einer sein) sitzt auf Zeile 21, mit dem rechten Rand in Spalte 8. Wenn wir ihn so zeichnen wollen, daß er auf Zeile a sitzt und der linke Rand sich in Spalte b befindet, numerieren wir die Zeilen und Spalten im Bild einfach um, damit dieses Ergebnis erzielt wird (siehe Abbildung 12.3).

Davon können wir das neue Programm ablesen:

```
1Ø  PRINT AT a-2, b+2; "███▀▀▀▀"
2Ø  PRINT AT a-1, b; "███████"
3Ø  PRINT AT a, b; "▐00000▌"
```

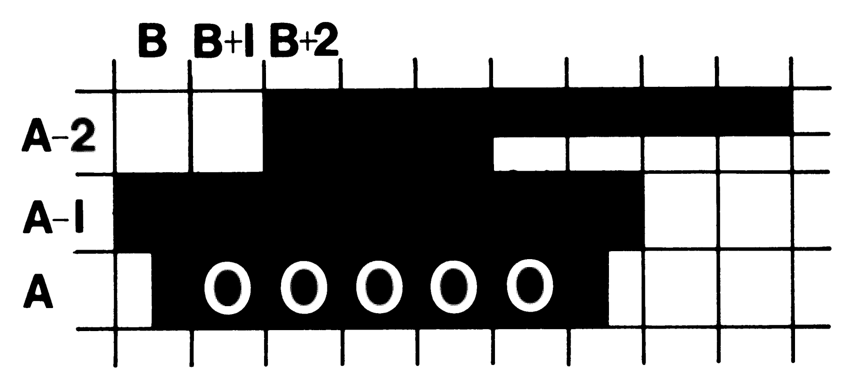


Abbildung 12.3

Natürlich läuft das nicht, wenn Sie den Wert von a und b nicht in einem vorangehenden Programmteil mitgeteilt haben. Es läuft auch nicht, wenn der Computer auf Werte von a und b stößt, die bewirken, daß das Bild über den Bildschirmrand hinausrutscht. Beim linken Rand bedeutet das, daß er Ø oder größer sein muß, auf der rechten Seite brauchen wir b+9 gleich 31 oder kleiner, demnach b = 22 oder kleiner. Ebenso muß a gleich 2 oder größer sein und 21 oder kleiner.

Der große Vorteil allgemeiner Überlegungen solcher Art ist der, daß wir unsere Panzer auf dem Bildschirm einfach dadurch zeichnen können, wo wir wollen, daß wir a und b bestimmen. Ein paar Proben: Fügen Sie sie den Zeilen 1Ø-3Ø oben an (die Zeilennummern werden vom Spectrum automatisch geholt) und fahren Sie das, damit Sie sehen, was dabei herauskommt.

a
```
  1 LET b = 7
  2 FOR a = 2 TO 21
 4Ø NEXT a
```

b
```
  1 LET a = 15
  2 FOR b = Ø TO 22
 4Ø NEXT b
```

c
```
  1 LET a = 2+15*RND
  2 LET b = 2Ø*RND
 4Ø GO TO 1
```

(Drücken Sie hier BREAK, um den Computer zu erlösen)

Sobald Sie sich mit Subroutinen (Unterprogrammen) auskennen, werden Sie für diese Fähigkeit, eine bestimmte Form an einer beliebig gewählten Stelle zu zeichnen, vielerlei Verwendung finden. Der nächste Abschnitt beschreibt eine häufig vorkommende Anwendung.

Bewegte Grafik

Nehmen wir an, Sie möchten den Panzer von links nach rechts über den Bildschirm laufen lassen. Programm b) erreicht das beinahe schon, hinterläßt allerdings einen Schwanz von Panzerhecks. Man *kann* sie durch Überschreiben von Leerstellen loswerden, aber am hübschesten geht es mit einem "unsichtbaren Rand", der das automatisch bewirkt. Verändern Sie die Zeilen 10-30 folgendermaßen:

```
 1  LET a = 15
 2  FOR b = Ø TO 21
1Ø  PRINT AT a−2, b+2; "□■■■▀▀▀▀"
2Ø  PRINT AT a−1, b; "□■■■■■■■"
3Ø  PRINT AT a. b; "□▐⓪⓪⓪⓪⓪▌□"
4Ø  NEXT b
```

Beachten Sie die Leerstellen, die vor jeder Grafikzeile neu hinzugekommen sind. (Um zu vermeiden, daß wir über den Rand laufen, lassen wir b in Zeile 2 nur bis 21 gehen.)

Der Rand aus Leerstellen ist auf dem Bildschirm nicht sichtbar. So, wie der Computer Zeichen anzeigt, heißt das aber, daß diese Leerstellen auf den unliebsamen "Schwanz" geschrieben werden und ihn damit löschen. (Nach einem alten Spruch fegen die Füchse ihre Spuren mit der Rute weg – das tut unser Panzer auch, nur ist sein Schweif unsichtbar.)

Wenn wir den Panzer in die entgegengesetzte Richtung laufen lassen wollen, brauchen wir auch am rechten Ende einen unsichtbaren Rand. Um den Panzer nach oben zu bewegen, ist ein Rand darunter erforderlich, soll er nach unten, ein Rand über ihm. Soll er sich in alle Richtungen bewegen (indem wir a und b entsprechend abändern), setzen wir den Rand um den ganzen Panzer, brauchen in den Reihen a−3 und a+1 also zwei Zeilen Leerstellen zusätzlich.

Aufgabe

Schreiben Sie ein Programm, mit dem der Panzer ständig um ein Bildschirmquadrat herumfährt, das die Größe 1Ø mal 1Ø hat. Setzen Sie rundherum unsichtbare Ränder; berechnen Sie, wie a und b sich verändern müssen; programmieren Sie das.

Bewegungssteuerung mit dem Keyboard

Da wir jetzt Dinge zeichnen können, die sich nach Wunsch bewegen, kann auch das Keyboard dazu verwendet werden, die Bewegungen von außerhalb des Programms zu steuern.

Auf plumpe Weise geht das, indem man das Programm als Schleife schreibt und über das Keyboard per INPUT etwas eingibt. Das hält alles, auch das bewegte Display, solange auf, bis die Tasten gedrückt werden.

Besser ist der INKEY$-Befehl. Eine Programmzeile

```
1Ø  LET c$ = INKEY$
```

befiehlt dem Computer, festzustellen, welche Taste gedrückt wird, und der Stringvariablen c$ das entsprechende Zeichen zuzuteilen. Wenn Sie mit diesem Zeichen geschickt umgehen, können Sie alles Mögliche anstellen.

Beispiel: Lenken wir den Panzer nach links oder rechts, wobei wir Taste 5 für links und Taste 8 für rechts verwenden. (Das ist wegen der Pfeile auf den Tasten eine Gedächtnisstütze für *Sie*, aber es ist nicht notwendig, CAPS SHIFT zu drücken und die Pfeile tatsächlich einzugeben.) Was machen wir? Wir weisen den Computer an, das INKEY$ zu lesen und die Anzeigeposition je nachdem zu verändern, ob 5 oder 8 gedrückt worden sind.
Also so:

```
 1  LET b = 15
 2  LET a = 12
 3  LET c$ = INKEY$
 6  IF c$ = "5" THEN LET b = b–1
 7  IF c$ = "8" THEN LET b = b+1
1Ø  PRINT AT a–2, b+2; □■■■▀▀▀▀□"
2Ø  PRINT AT a–1, b; "□■■■■■■■□"
3Ø  PRINT AT a, b; "□▐00000▌□"
4Ø  GO TO 3
```

Beachten Sie den unsichtbaren Rand an beiden Seitenkanten. Der Haken dabei: Wenn Sie die Tasten zu lange drücken, können Sie seitwärts vom Bildschirm rutschen. Versuchen Sie das Programm so abzuändern, daß das verhindert wird, indem Sie Zeilen einfügen à la IF a<Ø THEN . . . dieses oder jenes.

Solange Sie eine Taste gedrückt halten, geht das Programm weiter, und der Panzer bleibt in Bewegung. Manchmal ist das ausgesprochen ärgerlich; oft will man in Wahrheit, daß das Programm nur auf *Veränderungen* in INKEY$ oder wenigstens nur auf wiederholtes Drücken anspricht.

```
3  IF INKEY$ <> "" THEN GO TO 3
4  IF INKEY$ = "" THEN GO TO 4
5  LET C$ = INKEY$
   . . . . .
```

Das hat die Wirkung, daß bei Zeile 3 alles zum Stillstand kommt, wenn Sie immer noch die alte Taste gedrückt halten. Sobald Sie loslassen, geht der Computer zu 4 und bleibt dort stehen. Drücken Sie eine neue Taste oder erneut die alte – schon geht es weiter!

WARNUNG: Überlegen Sie sich genau, was der Computer Ihrem Befehl entsprechend mit diesem Inkey$ tun soll. *Sie* möchten vielleicht nur Tasten

drücken, die Zahlen liefern, um dan VAL INKEY$ zu bestimmen, mit dem die Zahl von einem Zeichen in etwas verwandelt wird, das man wirklich für arithmetische Berechnungen verwenden kann. Um aber das Programm starten zu können, müssen Sie ENTER drücken. Manchmal liest der Spectrum jedoch das als INKEY$ und versucht es in eine Zahl zu verwandeln – das Programm bricht zusammen. Oder noch schlimmer: Werden keine Tasten gedrückt, berechnet er VAL (den leeren String). Das kann etwas verwirrend sein, bis Sie begriffen haben. (Sie können sich durch die Verwendung geeigneter IF... THEN...-Befehle dagegen schützen.)

PRINT und PLOT kombiniert

In manchen Programmen müssen Sie für grafische Displays vielleicht PRINT und PLOT gemeinsam verwenden. Wichtig zu merken: PRINT AT x, y und PLOT x, y bewirken ganz verschiedene Dinge, weil die x, y sich auf zwei sehr verschiedene Koordinatensysteme auf dem Bildschirm beziehen. Das Handbuch von Sinclair zeigt im Kapitel 15 ein Schaubild, mit dem beide Systeme dargestellt werden. In der Regel lassen sich die beiden Befehle aber am einfachsten dadurch richtig verbinden, daß Sie auf kariertem Papier von dem Bereich, wo Sie Grafik zeichnen wollen, eine Rohskizze anfertigen. Zeichnen Sie beide Koordinatensysteme ein und ziehen Sie die Skizze zu Rate, während Sie Ihr Programm schreiben. Es lohnt oft, einige Zeit für das Nachdenken über den Programmaufbau aufzuwenden, *bevor* man sich ans Keyboard setzt. (Dagegen fällt es, wenn Ihr erster Versuch schiefgeht, oft leichter, die Fehler durch Experimentieren mit dem Computer auszumerzen, statt sich das Gehirn zu zermartern. Es kommt eben darauf an, Zeit und Mühe möglichst wirksam einzusetzen.)

PAUSE

Der Spectrum hat einen PAUSE-Befehl, der ihn veranlaßt, einen festgelegten Zeitraum zu warten. Bei bewegter Grafik ist das sehr nützlich, etwa dann, wenn man etwas verlangsamen möchte, das sonst zu schnell laufen würde. Wenn Sie eine Unterbrechung von n Sekunden erzielen wollen, tippen Sie

```
PAUSE 5Ø*n
```

13 BENUTZERGEWÄHLTE ZEICHEN

Wenn der Zeichenvorrat ein Zeichen, das Sie brauchen, nicht enthält, können Sie selbst ein neues erfinden.

Der Spectrum hat 21 Zeichen, die Sie nach Wunsch verändern können. Das sind im Zeichenvorrat die Nummern 144-164. Zunächst sind sie als die Buchstaben A-U ausgelegt. Um Zugang zu ihnen über die Tastatur zu erhalten, gehen Sie in den "G"-Modus und drücken die entsprechende Buchstabentaste.

Die Methode, Ihre eigenen Grafikzeichen zu entwickeln, ist im Grunde einfach, verdient aber eine angemessene Erläuterung, weil das eine wirklich elegante Sache ist und man damit aus einem langweiligen Programm etwas ganz Besonderes machen kann.

Der erste Schritt: Sie skizzieren das gewünschte Zeichen auf einem Gitter 8 mal 8 und schwärzen die Quadrate, die INK-Farbe haben werden, wenn das Zeichen angezeigt wird. Beispiel: Abbildung 13.1 zeigt ein "Katzen"-Zeichen aus dem Band "Computerrätsel: für Spectrum und ZX81" (erschienen bei Shiva).

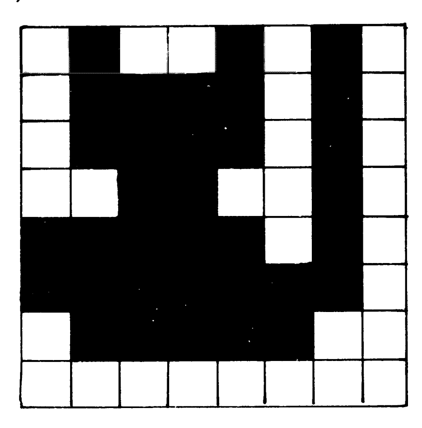


Abbildung 13.1

Ersetzen Sie nun die schwarzen Quadrate jeweils durch 1, die leeren jeweils durch Ø. Sie erhalten eine Liste dieser Art:

```
Ø  1  Ø  Ø  1  Ø  1  Ø
Ø  1  1  1  1  Ø  1  Ø
Ø  1  1  1  1  Ø  1  Ø
Ø  Ø  1  1  0  Ø  1  Ø
1  1  1  1  1  Ø  1  Ø
1  1  1  1  1  1  1  Ø
Ø  1  1  1  1  1  Ø  Ø
Ø  Ø  Ø  Ø  Ø  Ø  Ø  Ø
```

Anschließend bestimmen Sie, welchen Buchstaben Sie verwenden wollen. Der naheliegende ist "C".

Der mühsame (aber einfache) Weg besteht darin, über die Tastatur die folgenden Befehle einzugeben:

```
POKE USR "C", BIN Ø1ØØ1Ø1Ø
POKE USR "C" +1, BIN Ø1111Ø1Ø
POKE USR "C" +2, BIN Ø1111Ø1Ø
POKE USR "C" +3, BIN ØØ11ØØ1Ø
POKE USR "C" +4, BIN 11111Ø1Ø
POKE USR "C" +5, BIN 1111111Ø
POKE USR "C" +6, BIN Ø11111ØØ
POKE USR "C" +7, BIN ØØØØØØØØ
```

Betrachten Sie das, wenn Sie wollen, als ein Stück Zauberei! POKE stellt bestimmte Informationen in den Computerspeicher (siehe Kapitel 23 über PEEK und POKE), USR teilt ihm lediglich mit, daß usergewählte Zeichen in Aktion sind, BIN steht für "binär". Wichtig dabei sind das "C", also der Buchstabe, den Sie gewählt haben, die angefügten Zahlen 1, 2, . . . 7 und die Folgen von Ø und 1, die ihrerseits von der Tabelle nach meinem Katzenbild abgeschrieben sind.

Jedes 8x8-Muster von Nullen und Einsen kann so behandelt werden; der Computer speichert auf einmal bis zu 21 verschiedene benutzergewählte Zeichen. Sie werden durch NEW *nicht* gelöscht, mit SAVE aber auch nicht gesichert.

Es gibt noch andere Methoden, die Zeichen zu erzeugen. Ich habe ein Fertigprogramm geschrieben (ZEICHENKONSTRUKTEUR), mit dem Sie Ihr Zeichen *konstruieren* und es dann laden können. Ein Weg ist der, die Binär- in Dezimalzahlen zu verwandeln – etwa durch direkte Befehle wie

```
PRINT BIN Ø1ØØ1Ø1Ø
```

was die Zahl 74 ergibt. Die Reihen der Katze in Dezimalzahlen lauten

74, 122, 122, 5Ø, 25Ø, 254, 124, Ø

Statt mit POKE in den BIN herumzustochern, können Sie diese Zahlen auch direkt verwenden:

```
POKE USR "C", 74
POKE USR "C" +1, 122
. . . . .
POKE USR "C" +7, Ø
```

Wenn Sie diese Anweisungen ins Programm schreiben, können Sie die Zeichen bei einem Lauf mit RUN aufbauen, und wenn Sie das Programm sichern, werden auch die Zeichen gesichert.

Sie können die Zeichen auch damit sichern, daß Sie eine Abart von SAVE verwenden, die einen Speicherblock sichert, und auf dieselbe Weise mit LOAD zurückladen. Für diesen Band hier geht das aber ein bißchen zu weit.

Selbstverständlich können Sie die Liste der Zahlen 74, 122, etc. als ein Array eingeben oder den DATA-Befehl verwenden (siehe Kapitel 18 über Daten.)

Aufgaben

1 Bauen Sie benutzergewählte Zeichen für die vier Skatkartenfarben auf, gespeichert als "H", "T", "K" und "P". Verwenden Sie dazu die Gitter von Abbildung 13.2.

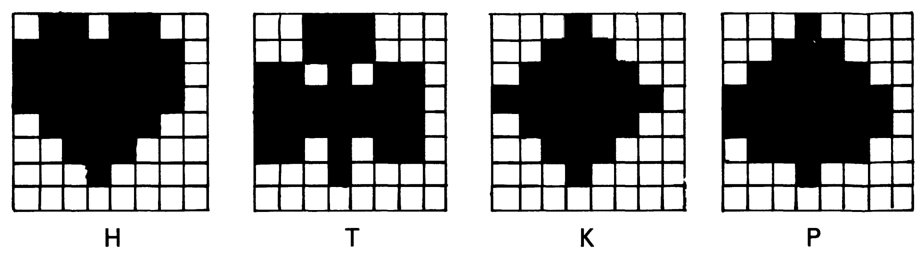


Abbildung 13.2

2 Benützen Sie Abbildung 13.3 dazu, ein Zeichen für "1/2" aufzubauen.

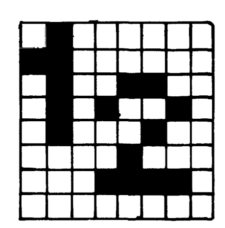


Abbildung 13.3

3 Entwerfen Sie einen Satz Zeichen für die zwölf Tierkreiszeichen und bauen Sie sie in den Positionen A-L auf.

4 Entwerfen Sie zwei verschiedene Zeichen, die einen Hund darstellen und sich nur durch die Schwanzstellung unterscheiden. Zeigen Sie sie mit PRINT abwechselnd an einem festen Platz an und sorgen Sie dafür, daß der Hund mit dem Schwanz wedelt.

5 Machen Sie einen gehenden Menschen, vielleicht so, daß Sie mehrere aneinandergefügte Zeichen verwenden.

14 SUBROUTINEN (Unterprogramme)

Wenn Sie eine Aufgabe in eine Anzahl unterscheidbarer Unteraufgaben zerlegen können, wird eine mächtige Waffe im Arsenal des Programmierers zugänglich: Subroutinen (Unterprogramme).

Es gibt eine Geschichte über Mathematiker, die ebenso für Programmierer gelten könnte (ja, es *sind* zwei verschiedene Lebewesen!). Sie hat damit zu tun, wie man Teewasser aufsetzt. Der Mathematiker wird zunächst aufgefordert, die einzelnen Schritte zu beschreiben, die zu einer fertigen Tasse Tee führen, wenn der Teekessel an einem Wandhaken hängt. Er sagt dann etwa: "Nimm den Teekessel vom Haken, füll ihn mit Wasser, stell ihn auf den Herd, zünde ein Streichholz an . . ." und so weiter. Danach soll er die Schritte schildern, die zu einer fertigen Tasse führen, wenn der Teekessel auf dem Küchentisch steht. Er antwortet: "Häng den Teekessel an den Haken und verfahre wie vorher."

Was macht er da? Er verwendet den ersten Ablauf als eine *Subroutine* (ein Unterprogramm). Beim zweitenmal verändert er die bestehende Lage so, daß die Subroutine anwendbar wird, und verwendet sie dann so, als sei ein einziger, unteilbarer Ablauf gegeben.

In der Computersprache ist eine Subroutine ein Programmblock, der selbständig geschrieben und wiederholt als Bestandteil eines größeren Programms verwendet werden kann. Subroutinen stellen eine sehr verfeinerte Art dar, Programme zu schreiben, weil man mit ihnen im Regelfall leichter erkennen kann, was vorgeht. Außerdem läßt sich ein Programm, das mit Subroutine geschrieben ist, leichter von Fehlern befreien, weil man jedes für sich "entfehlern" kann und dann nur noch zu prüfen braucht, ob sie richtig miteinander verbunden sind.

Der dafür zuständige Befehl ist GO SUB. Das ist ähnlich wie GO TO, aber beträchtlich vielseitiger. Er tritt in typischer Art folgendermaßen auf:

```
100  GO SUB 500
110  anderes Zeug
     .....
500  tu etwas
     .....
570  RETURN
```

wobei, wie üblich, der kleingedruckte Text andere Programmteile bezeichnet. Bewirkt wird dadurch Folgendes:

a Sobald das Programm auf Zeile 100 stößt, springt es zu 500 und *merkt sich, wo es hergekommen ist.*

b Es führt dann 5ØØ und alles Folgende aus, bis es zu Zeile 57Ø kommt und einen RETURN-Befehl erhält.

c Es kehrt dann zur Ursprungszeile (hier 1ØØ) zurück und führt die *nächstfolgende* Anweisung aus.

Im Grunde ist das wie GO TO, nur findet die Rückkehr zum Ausgangspunkt automatisch statt. Die Haupteigenschaft ist aber die, daß man einunddieselbe Subroutine von *verschiedenen* Zeilen im selben Programm aus anspringen kann. Der Computer behält im Gedächtnis, wo er eingestiegen ist und kehrt mit RETURN wieder dorthin zurück.

Als Beispiel zeige ich Schritt für Schritt, wie ein Programm geschrieben wird, das

d die Tasten 5, 6, 7, 8 dazu verwendet, auf dem Bildschirm einen Cursor [P] zu bewegen,

e anstelle des Cursors jede Zeicheneingabe vom Keyboard anzeigt.

Für diese Zeicheneingabe müssen wir INKEY$ verwenden. Wir lesen damit die jeweils gedrückte Taste und teilen sie einer Stringvariable a$ zu. Wir wollen, daß das Programm nur auf neu gedrückte Tasten reagiert, wie das auf Seite 65 beschrieben ist, benötigen also einen Programmblock in der Form

```
1ØØØ  IF INKEY$ <> "" THEN GO TO 1ØØØ  ┐
1Ø1Ø  IF INKEY$ = "" THEN GO TO 1Ø1Ø   ├ wird die Subroutine
1Ø2Ø  LET a$ = INKEY$                  ┘
```

Die 1ØØØer-Zeilennummern werden verwendet, weil das eine Subroutine sein wird und wir sie weit wegstellen wollen (allerdings kann man oft einige Programmzeilen sparen, wenn man alle Subroutinen an den *Anfang* stellt und den Lauf mit GO TO und nicht mit RUN beginnt – aber das ist eine Verfeinerung, auf die hier einzugehen nicht lohnt). Um wieder herauszukommen, brauchen wir die zusätzliche Zeile

```
1Ø3Ø RETURN
```

Um diesen [P]-Cursor bewegen zu können, brauchen wir als nächstes . . . was? Auf jeden Fall müssen wir ja einmal wissen, *wo* er angezeigt werden soll. Wir erfinden also zwei Variable a und b für die Werte von Reihe und Spalte, wo mit PRINT angezeigt werden soll. Damit das Ganze nicht zusammenbricht, bevor wir richtig angefangen haben, müssen wir ihnen Werte zuteilen. Die Bildschirmmitte ist ein guter Ausgangspunkt.

```
1Ø  LET a = 1Ø
2Ø  LET b = 15
```

Nun wollen wir die Tasten 5, 6, 7, 8 dazu verwenden, a und b zu verändern, um damit den Cursor zu bewegen. Die brave Subroutine oben wird die Tastatur lesen, also ergibt sich naheliegenderweise als nächstes

```
3Ø  GO SUB 1ØØØ
```

Nun wird a$ uns mitteilen, welche Taste von 5, 6, 7, 8 gedrückt worden ist. Wir wollen den Cursor [P] in der Richtung der vier Pfeile auf diesen Tasten bewegen. (Deshalb verwenden wir sie auch – die Pfeile sind gute Gedächtnisstützen!)

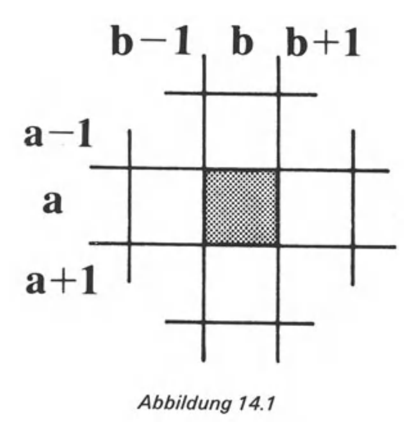


Abbildung 14.1

Sehen Sie sich Abbildung 14.1 an. Sie zeigt Position a, b und die vier Nachbarn. Wenn wir die Hinweise im Bild nutzen, sehen wir, was wir haben wollen:

Taste 5 soll b zu b −1 verändern und a in Ruhe lassen,

Taste 6 soll a zu a +1 verändern und b in Ruhe lassen,

Taste 7 soll a zu a −1 verändern und b in Ruhe lassen,

Taste 8 soll b zu b +1 verändern und a in Ruhe lassen.

Hier ein Weg:

```
4Ø   IF a$ = "5" THEN LET b = b−1
5Ø   IF a$ = "6" THEN LET a = a+1
6Ø   IF a$ = "7" THEN LET a = a−1
7Ø   IF a$ = "8" THEN LET b = b+1
```

Nachdem wir den Cursor bewegt haben, möchten wir sehen, wo er hingegangen ist; spaßeshalber lassen wir ihn blinken:

```
8Ø   PRINT AT a, b; FLASH 1; "P"
```

Und weiter? Wir wollen anstelle dieses [P] ein Zeichen eingeben, das angezeigt werden soll. Wieder die Subroutine!

```
9Ø   GO SUB 1ØØØ
```

Diesmal liest der Computer das Keyboard, teilt a$ den Wert zu, den er dort findet (jeweils das, was wir gedrückt haben, a, b, c, d, . . ., 7, 3, >, . . .) und kehrt zur *nächsten Zeile nach Zeile 90 zurück*, Um ihm zu sagen, daß er das gefundene Zeichen anzeigen soll, müssen wir noch anfügen

```
1ØØ  PRINT AT a, b; a$
```

Fast fertig. Bis jetzt läuft das Ganze nur einmal. Wir wollen wieder zum Anfang zurück und neu beginnen – das neue a, b aber behalten und es *nicht* auf 1Ø, 15 zurücksetzen. Das ist einfach:

```
11Ø  GO TO 3Ø
```

(3Ø schickt ihn dann sofort wieder zu 1ØØØ zurück. Warum funktioniert 11Ø GO SUB 1ØØØ nicht?)

Geben Sie alle oben aufgeführten Zeilen ein (wir haben dafür gesorgt, daß die Nummern in der richtigen Reihenfolge sind, was nicht immer gleich beim erstenmal der Fall ist, wenn man ein Programm schreibt; lassen Sie sich also nicht irreführen!) und drücken Sie RUN. Passieren wird gar nichts, aber wenn Sie 5, 6, 7 oder 8 drücken, werden Sie den [P]-Cursor an seiner neuen Position sehen. (Wenn Sie irgend etwas anderes drücken, erscheint er beim alten a, b – ein Programmvorteil, an den wir gar nicht gedacht hatten.) Drücken Sie dann eine Taste, meinetwegen t. Das [P] verschwindet und macht einem t Platz. Bewegen Sie den (jetzt unsichtbaren) Cursor mit 5, 6, 7, 8; zeigen Sie das nächste Zeichen an und machen Sie so weiter. Sie können den ganzen Bildschirm mit Wörtern beschreiben. (Wie wäre es mit einem Programm für computerunterstützte Kreuzworträtsel-Konstruktion?)

Damit das Programm an den Bildschirmrändern nicht zusammenbricht, ist ein klein wenig Userschutz von Nutzen:

```
75  IF a<Ø OR a>21 OR b<Ø OR b>31 THEN GO TO 3Ø
```

Ihnen fallen sicher noch mehr Verbesserungen ein.

Aufgabe

Generieren Sie statt der Subroutine INKEY$ sowohl Eingaben mit 5, 6, 7, 8 als auch die mit PRINT anzuzeigenden Zeichen per Zufall; setzen Sie das Ganze in Gang und warten Sie die Entwicklung ab. Verwenden Sie, um Zufallszeichen zu erhalten, Befehle wie PRINT CHR$ INT (65 +26 * RND), was zufällig aus dem Alphabet auswählt. (Warum?)

Hier noch ein Beispiel – eine Einführung in die Computer-Kunst. Gezeichnet werden Zufallsquadrate, schwarz oder kariert, bis der Speicherplatz zu Ende geht oder Sie mit BREAK unterbrechen.

```
1Ø  LET a = 1Ø * RND
2Ø  LET b = 1Ø * RND
3Ø  LET q = 5 * RND
```

```
  4Ø  LET r = 5 * RND
  5Ø  LET k = INT (2 * RND)
  6Ø  IF k = Ø THEN LET m$ = "■"
  7Ø  IF k = 1 THEN LET m$ = "▚"
  8Ø  GO SUB 1ØØØ
  9Ø  GO TO 1Ø
1ØØØ  FOR i = a TO a +q
1Ø1Ø  FOR j = b TO b +r
1Ø2Ø  PRINT AT i, j; m$
1Ø3Ø  NEXT j
1Ø4Ø  NEXT i
1Ø5Ø  RETURN
```

Aufgabe

Generieren Sie eine Zufallszahl zwischen 1 und 6 mit INT (1 +6 * RND). Zeigen Sie je nachdem, wie diese Zahl, sagen wir n, lautet, auf dem Bildschirm bei der Mitte n Punkte in genau der Anordnung an, wie sie auf einem Würfel erscheinen. Wiederholen Sie das, wenn ENTER gedrückt wird.

Erfinden Sie dazu eine Subroutine für jede der sechs Möglichkeiten. Wenn die Subroutine für n ab (meinetwegen) Zeile 5ØØ +1ØØ * n geschrieben wird (das heißt also, bei 6ØØ, 7ØØ und so weiter) und deutlich vor dem Beginn der nächsten endet, können Sie schwindeln und GO SUB 5ØØ +1ØØ * n verwenden, um komplizierte bedingte Sprünge zu vermeiden. Sie brauchen aber auf jeden Fall sechs getrennte RETURN-Befehle . . .

Verwenden Sie für die Wiederholung mit ENTER eine Zeile, die mit INPUT k$ ein Eingabezeichen verlangt; setzen Sie fort mit einem GO TO-Befehl, der alles wieder zum Anfang zurückschickt. k$ wird für gar nichts verwendet, aber der Computer wartet darauf, erhält ENTER und geht weiter zum GO TO. Alles klar?

15 SON ET LUMIÈRE

Der Spectrum singt den Blues.

Probieren Sie Folgendes:

```
10  FOR i = 1 TO 6
20  INK i
30  CIRCLE 100, 80, i * 10
40  BEEP 0.5, i
50  NEXT i
```

Das mag nicht sonderlich aufregend sein, veranschaulicht aber ein paar Punkte. Erstens läßt sich erkennen, wie wir die Farbe des Symbols verändern können, das grafisch gesetzt oder mit PRINT angezeigt wird. Wir geben einfach die mit der gewünschten Farbe verbundene Zahl in einer INK-Anweisung an. Wir brauchen uns nicht zu merken, welche Zahl welcher Farbe entspricht, weil die Farben über den entsprechenden Ziffern auf dem Keyboard angegeben sind. Im obigen Programm wird also beim erstenmal Zeile 10 mit i = 1 ausgeführt; die Anweisung "INK 1" wird von BASIC verstanden als "verwende bis auf weiteres für Anzeigen und Zeichnen blaue Tinte". Das einzige, was gezeichnet wird, solange blaue "Tinte" Verwendung findet, ist freilich der kleinste Kreis. Bis der nächste Kreis gezeichet wird, ist Zeile 20 zu "INK 2" geworden, also rot, und so weiter.

Sobald ein Kreis fertig ist, läßt der Computer ein triumphierendes Piepen hören. Das ist die Folge von Zeile 40. Der erste Wert nach BEEP liefert die Dauer des Tons in Sekunden (in diesem Programm dauert also jeder 0.5 Sekunden, obwohl sich das länger anhört, nicht?), der zweite bestimmt den Ton, der gespielt werden soll. Ist dieser Wert Null, dann ist der Ton das mittlere C, 1 ist C#, 2 ist D, 3 ist D#, 4 ist E, 5 ist F (einen Ton E# gibt es nicht!), und so weiter. Negative Werte führen unter das mittlere C. So ist −1 B, −2 A#, etc.

Die einfachste Verwendung von BEEP ist die, dem User anzuzeigen, daß etwas geschehen ist (etwa ein Fehler im Programm) oder daß der Computer auf Eingabe wartet oder sie angenommen hat. Natürlich kann man damit auch Musik machen, aber das möchte ich auf später verschieben (siehe Fertigprogramme) und mich hier mit anderen Eigenschaften des Farbensystems befassen.

Erstens läßt sich nicht nur die Inkfarbe verändern. Der Hintergrund (den BASIC recht sinnvoll PAPER, also Papier nennt) kann acht verschiedene Farben haben. Wenn INK und PAPER von derselben Farbe sind, sehen Sie natürlich gar nichts, was Verwirrung stiften kann, wenn Sie LIST drücken, nachdem ein Programm gelaufen ist, das die Farbe von INK zu der von PAPER verändert. (Ich erwähne das deshalb, weil mir das immer wieder unterläuft und ich mich dann frage, weshalb eigentlich das Programm gelöscht worden ist. Das ist natürlich gar nicht der Fall, ich kann es nur nicht sehen.)

Sie erwarten jetzt vielleicht, daß Sie "PAPER 2" eingeben können und einen roten Hintergrund erhalten, aber wenn Sie diese Anweisung meinetwegen in Zeile 5 in das Programm einschalten, passiert, sobald Sie RUN gedrückt haben, gar nichts. Der Grund: Das System kann die Farbe von PAPER, das schon beschrieben worden ist, nicht verändern, und es kann nicht sicher sein, daß nichts geschrieben worden ist, bis es einen CLS (clear screen)-Befehl ausgeführt hat. Der Computer reagiert also erst bei der Begegnung mit CLS auf den neuesten PAPER-Befehl.

Man kann ferner die Randfarbe verändern (beispielsweise, wenn man BORDER 4 eingibt, um einen grünen Rand zu erhalten), und das gilt von dem Augenblick an, in dem der Befehl auftaucht. Wenn Sie also einfügen

```
25  BORDER i
```

wird der Rand stets dem gezeichneten Kreis entsprechen.

Verändern Sie Zeile 25 so:

```
25  BORDER 7 -i
```

Das ist ein bißchen auffälliger (um nicht zu sagen: geschmacklos), nicht wahr?

Fügen Sie jetzt folgende Zeilen an:

```
 6Ø  INK 4
 7Ø  PLOT 1ØØ, Ø
 8Ø  DRAW Ø, 175
 9Ø  PLOT Ø, 8Ø
1ØØ  DRAW 255, Ø
```

und drücken Sie RUN.

Wie Sie erwartet haben, wird ein grünes Fadenkreuz eingezeichnet, aber wenn Sie genau hingucken, werden Sie sehen, daß alle Kreise in dem Bereich, wo das Fadenkreuz sie schneidet, grün gefärbt worden sind. Das ist kein Defekt Ihres Fernsehapparats oder eine Macke im Spectrum, sondern eine Eigenheit der Art, wie der Spectrum das Display behandelt. (In der Computersprache wird ein Fehler, den man nicht beheben kann, stets als "Eigenheit" bezeichnet, damit der Eindruck erweckt wird, das gehöre so.) Der Grund für dieses seltsame Verhalten ist jedenfalls der, daß die Attribute eines Punktes auf dem Bildschirm (also seine Farbe, die Helligkeit, ob er blinkt) nicht auf nur ein Pixel beschränkt sind. Sie beziehen sich auf ein ganzes Zeichen, das, wie wir bereits gesehen haben, 64 Pixel in einem Quadrat 8 × 8 belegt. Wenn Sie die Farbe eines Pixels verändern, wechseln alle anderen vom selben 8 × 8-Bereich die Farbe.

In den meisten Fällen ist das erträglich. In der Regel treten auch keine Probleme auf, wenn man die Helligkeit eines Zeichenquadrats verändert. Das geschieht durch die Anweisung BRIGHT 1, um die Helligkeit zu steigern, und BRIGHT Ø, um sie wieder zu verringern. Sobald ein BRIGHT 1-Befehl ausgeführt worden ist, wird alles hell gezeichnet, bis das nächste BRIGHT Ø auftaucht. Wollten wir also die zwei inneren Kreise heller haben als die anderen, könnten wir die Anweisungen hinzufügen

```
5   BRIGHT 1
23  IF i>2 THEN BRIGHT Ø
```

Blinken ist eine ganz andere Sache. Sie können Blinken zu- und abschalten durch FLASH 1 und FLASH Ø, ganz ähnlich wie beim Umgang mit BRIGHT. Falls Sie das aber beim Programm für das Zeichnen der Kreise versuchen, blinken wegen dem Problem mit den Attributen große Bildschirmblöcke Sie auf recht entnervende Weise an. Probieren Sie

```
4  FLASH 1
6  CLS
```

um zu sehen, was ich meine. (Wie PAPER braucht auch FLASH ein CLS, um in Aktion treten zu können.) Mag sein, daß Sie für ein solches Display Verwendung haben. Die einzige, die ich mir vorstellen kann, ist die, Migräne bei Laborratten hervorzurufen.

Allgemein wird FLASH also vorwiegend in PRINT-Anweisungen verwendet, wo wir ganze Zeichenquadrate blinken lassen wollen, statt bei PLOT, DRAW und CIRCLE, wo die Effekte eher unerwünscht sein dürften.

Noch besser: Alle Attribute, INK, PAPER, FLASH und so weiter können "eingebettet" werden in einen PRINT-Befehl. Ihre Wirkung wird dadurch auf die Symbole begrenzt, die mit dieser Anweisung angezeigt werden; außerdem brauchen Sie kein CLS, um sie wirksam zu machen. Beispielsweise können Sie schreiben

```
11Ø  PRINT AT 1Ø, Ø; INK 5; FLASH 1; PAPER 4; "xxx"
```

und RUN geben (nachdem Sie Zeile 4 gelöscht und FLASH Ø: CLS als direkten Befehl eingegeben haben, um das Blinken loszuwerden), so daß nur die x blinken. Wenn Sie jetzt LIST drücken, werden Sie sehen, daß (wegen Zeile 6Ø) INK immer noch grün ist, obwohl sie in 11Ø auf hellblau gesetzt wurde, und das Listing nicht blinkt.

Ich habe mich nicht mit allen Möglichkeiten der Displaybehandlung befaßt, die der Spectrum zuläßt, und werde das auch nicht tun. Sie haben einen Einführungsband vor sich. Sein ganzer Sinn besteht eben darin, Sie nicht mit zu vielen Einzelheiten zu belasten. Wenn Sie diese Methoden beherrschen, werden Sie natürlich den Rest aus dem Handbuch erlernen wollen. Das dürfte dann nicht allzuschwer fallen.

Zum Abschluß noch drei kleine Punkte. Erstens: Daß der Bildschirm geleert werden muß, bevor manche Attribute ihre Wirkung entfalten, heißt nicht, daß Sie unbedingt CLS eingeben müssen. Beispielsweise hat das Drücken der Taste ENTER ohne vorherige Eingabe dieselbe Wirkung wie CLS: LIST. Statt also FLASH: CLS zu tippen, um das Blinken zu unterbinden, können Sie FLASH Ø geben und dann ENTER statt nur einmal gleich zweimal drücken.

Zweitens: Es wird Ihnen aufgefallen sein, daß der Randbereich knapp unter dem Paper sonderbar reagiert, sich nicht immer verändert, wenn er das tun sollte, sondern seine Farbe am Beginn eines Laufs beibehält. Vielleicht ist Ihnen klargeworden, daß das die Befehls- und Meldezeile ist (also der Platz, wo das BASIC-System mit Ihnen Verbindung hält). Da liegt der Hund begraben: Das BASIC-System will nicht in die vorhin erwähnte Falle tappen, mit Ink grün auf

Paper grün zu schreiben. Schließlich ist es ja sehr wichtig, daß Sie erfahren, wann Ihnen etwas mitgeteilt werden soll. Tatsächlich wird je nach Randfarbe zwischen Schwarz und Weiß gewechselt, damit das Resultat so deutlich wie möglich ausfällt. Dieser Bereich hat also bei jedem Lauf eine feste Farbe. Wenn Sie das ganze Programm durch Einfügen von

```
120 RUN
```

in eine Schleife setzen und an verschiedenen Stellen mit BREAK hineingehen und wieder CONT drücken, werden Sie die Wirkung ganz deutlich erkennen.

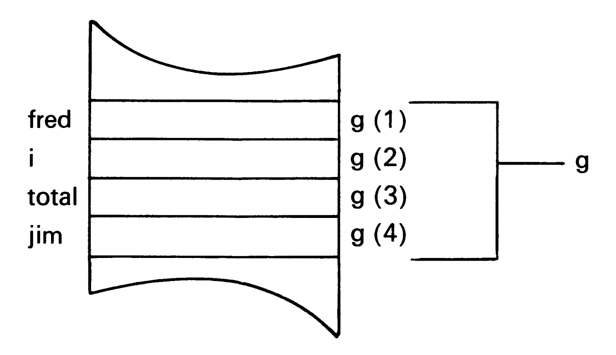
Drittens eine Stilfrage: Wenn Sie "FLASH 1" schreiben, schalten Sie die Blinkwirkung zu. Viel schöner wäre doch, wenn Sie schreiben könnten, "FLASH an" und "FLASH aus" oder "BRIGHT an" und "BRIGHT aus", statt das doch recht sinnlose Ø und 1 zu verwenden. Das geht ganz leicht. Fügen Sie eine Zeile 1 an:

```
1 LET an = 1 : LET aus = Ø
```

Wenn Sie jetzt "FLASH an" schreiben, ersetzt BASIC 1 durch "an" (oder Ø durch "aus"), und das Programm liest sich besser.

Arrays

Man kann eine Gruppe von Speicherzellen zusammenfassen, indem man ihnen denselben Namen gibt. Eine solche Gruppe wird Array genannt (deutsch etwa: Feld, Tabelle). Wir können uns das so vorstellen: Hier ist ein Speicherblock:



Im Normalfall würden wir jeder Zeile einen Namen geben (fred, i, etc.), wie links zu sehen. Wir können aber alle vier Zellen beim gleichen Namen nennen (ich habe "g" gewählt), wie Sie rechts erkennen. Wenn wir diesen Weg wählen (und warum wir das wollen, wird nicht auf Anhieb klar sein), muß dreierlei berücksichtigt werden:

1. Ein Arrayname ist stets ein einzelner Buchstabe, also können Sie ein Array nicht t2 oder albert nennen.
2. Arrays können so lang sein, wie Sie wollen (innerhalb der Grenzen, die der Speicher des Geräts setzt). Sie müssen BASIC also sagen, wie groß

jedes ist, bevor Sie es verwenden. Dafür gibt es eine DIM-Anweisung (DIM für dimensionieren). Sie sähe beim Array im Diagramm so aus:

```
10 DIM g(4)
```

Mit anderen Worten: Der Name des Arrays wird zusammen mit der Zahl seiner Zellen in Klammern angegeben.

3 Wir müssen einzelne Zellen innerhalb des Array unterscheiden können. Wie das gemacht wird, ergibt sich aus dem Diagramm; die erste Zelle wird als g(1) bezeichnet, die zweite als g(2) und so weiter.

Sehen wir uns an, wie eine Folge von 2Ø Zahlen in ein Array gesetzt werden könnte. Am einfachsten wäre die Schreibweise:

```
  1  DIM n(2Ø)
  2  INPUT n(1)
 3Ø  INPUT n(2)
 4Ø  INPUT n(3)
     .....
21Ø  INPUT n(2Ø)
```

Das ist aber ersichtlich mühsam und besitzt, wie deutlich erkennbar, keinen Vorteil gegenüber der Verwendung einzelner Variablennamen, weil in den Anweisungen von 2 bis 21Ø jedes Arrayelement eigens benannt wird.

Der Kniff dabei ist aber der, daß der Klammerinhalt keine Zahl zu sein braucht. Wir können also zum Beispiel von n (z) sprechen. Das bedeutet n (2), wenn z = 2 und n (17), wenn z = 17.

Jetzt ist das Problem leicht zu lösen. Wir sehen, daß der Wert in Klammern in Einerschritten von 1 zu 2Ø geht – das Stichwort für eine FOR-Schleife:

```
1Ø  DIM n (2Ø)
2Ø  FOR z = 1 TO 2Ø
3Ø  INPUT n (z)
4Ø  NEXT z
```

Beim ersten Durchgang durch die Schleife ist z = 1, also entspricht Zeile 3Ø dem Befehl INPUT n (1). Beim zweitenmal ist z = 2, Zeile 3Ø demnach INPUT n (2), und so weiter. Ich habe den Variablennamen "z" ganz bewußt gewählt. Er *zeigt* nämlich die ganze Zeit auf das jeweilige Arrayelement, eine sehr nützliche Methode für den Umgang mit Arrays.

CHR$ mit Musik

Mit Arrays wollen wir jetzt etwas Interessantes anstellen. BEEP haben wir schon kennengelernt und erwähnt, man könne damit Musik machen, obschon das auf der niedrigsten Ebene mühselig wäre, weil wir für jeden Ton die Zahl errechnen und dann eine ganze Reihe von BEEP-Befehlen mit den entsprechenden Zahlen eingeben müßten.

Warum soll nicht der Computer die Arbeit tun und die Umrechnung von Tönen in Zahlen für uns übernehmen?

Wir wollen am Anfang beim Einfachen bleiben und nur in einer Oktave arbeiten, nämlich in der um das mittlere C. Die Zahlencodes für diese Töne sind:

A : –3
A# : –2
B : –1
C : Ø
C# : 1
D : 2

D# : 3
E : 4
F : 5
F# : 6
G : 7
G# : 8

Da die erhöhten Töne stets einen Wert höher liegen als der entsprechende Ton ohne Vorzeichen, brauchen wir keine Tabelle. Die erforderliche Grundinformation sieht also so aus:

A : –3
B : –1
C : Ø
D : 2
E : 4
F : 5
G : 7

Ein klarer Zusammenhang zwischen diesen Zahlen besteht nicht, also empfiehlt es sich, sie in ein Array aufzunehmen und herauszusuchen, sobald wir sie brauchen.

Sie zu speichern, ist einfach; wir könnten schreiben:

```
1Ø  DIM s (7)
2Ø  LET s (1) = −3: LET s (2) = −1: LET s (3) = Ø: LET s (4) = 2
3Ø  LET s (5) = 4: LET s (6) = 5: LET s (7) = 7
```

(Es gibt andere Wege, die befriedigender wären, wenn wir es mit einem größeren Array zu tun hätten, aber hier funktioniert das sehr gut.)

Wir geben nun einen Ton (als Buchstaben) über das Keyboard ein, müssen ihn dann aber dazu benützen können, die entsprechende Zahl im Array s "nachzuschlagen". Mit anderen Worten: Wir brauchen etwa:

```
40  INPUT  "Eingeben Ton"; n$
           [Ein Prozeß, der den richtigen Wert von p aus dem von n$
           errechnet (z. B. wenn n$ = "A", z = 1, wenn n$ = "B",
           z = 2, etc.)]

100  BEEP 0.5, s (z)
```

Wir könnten jetzt natürlich eine Folge von Anweisungen schreiben wie:

```
45  IF n$ "A" THEN LET z = 1

50  IF n$ "B" THEN LET z = 2

    .....
```

und das würde auch gehen, aber es ist umständlich und unschön. Ein besserer Weg: Man verwendet die internen numerischen Codes der Zeichen. "A" wird danach gespeichert als die Zahl 65, "B" als 66, und so weiter. (Siehe Anhang A des Handbuchs).

Um den Code für A zu erreichen, können Sie schreiben:

```
LET z = CODE "A"
```

Wenn wir also schreiben:

```
50  LET z = CODE n$
```

würden wir für z Werte von 65, 66, 67 erhalten, die offenkundig um 64 größer sind, als wir sie haben wollen. Zeile 50 müßte also lauten:

```
50  LET z = CODE n$-64
```

Fügen Sie nun noch eine Zeile an, um zur Eingabeanweisung zurückzukehren:

```
110  GO TO 40
```

und wir haben eine (sehr) primitive Klaviatur. (Vergessen Sie nicht, bei der Eingabe der Töne Großbuchstaben zu verwenden!)

Als erstes krankt das daran, daß wir, um es uns leicht zu machen, auf die Halbtöne verzichtet haben. Wenn Sie also "C#" eingeben, erhalten Sie doch C. Da die CODE-Funktion den Code des ersten Buchstabens in einem String behandelt, wird das "#" nicht einmal wahrgenommen!

Wir müssen uns das zweite Symbol im String vornehmen, n$ (2)

```
60  IF n$ (2) = "#" THEN LET i = 1
```

Dann können wir Zeile 1ØØ abändern zu

```
100  BEEP Ø.5, s (z) +i
```

so daß der gespielte Ton dann, wenn i = 1, um 1 erhöht wird. Selbstverständlich muß i am Anfang der Schleife auf Null gesetzt werden, weil i sonst, sobald der erste Halbton auftaucht, bei 1 stehenbleibt und alle Töne als Halbtöne behandelt werden:

```
45  LET i = Ø
```

Leider wird die Sache dadurch ein wenig kompliziert, daß n$ (2) nicht existiert, wenn Sie einen Ton ohne Vorzeichen (etwa nur B) eingeben. So, wie die Dinge stehen, läuft das Programm also, wenn Sie C#, A#, G# eingeben, sobald Sie es aber mit C versuchen, erhalten Sie eine Fehlermeldung.

Der einfache Ausweg aus dieser Schwierigkeit ist der, n$ als ein Array von der Länge 2 festzulegen:

```
11  DIM n$ (2)
```

Jetzt existiert n$ (2) immer; wird es nicht verwendet, so besetzt BASIC es mit einer Leerstelle.

Das zweite, was bei unserem Musikinstrument nicht stimmt, ist, daß es Töne stockend spielt, je nach Ihrem Tippvermögen und den Eigenheiten der Tastatur (schließlich soll der Spectrum ja keine elektronische Orgel sein.)

Statt einen Ton gleich zu spielen, wenn er eingegeben wird, sollten Sie ihn speichern (erraten – in einem Array), und die ganze Melodie dann spielen, wenn alle Töne im Computer sind. Das hat den zusätzlichen Vorteil, daß wir die Melodie jederzeit wiederholen können.

Als erstes brauchen wir ein weiteres Array für die codierten Töne:

```
12  DIM t (15ØØ)
```

Unsere Melodie kann bis zu 15ØØ Töne enthalten.

Die Eingabeschleife verändert sich, weil die Zahl der Töne nicht mehr unbegrenzt ist, und auch deshalb, weil der Wert, der mit BEEP wiedergegeben werden soll, in t gespeichert und nicht gespielt wird. Ferner müssen wir aus der Schleife wieder herauskommen, falls es insgesamt weniger als 15ØØ Töne werden.

Die Zeilen 2Ø und 3Ø verändern sich nicht und stehen natürlich außerhalb der Schleife, also beginnen wir bei Zeile 35:

```
35  FOR q = 1 TO 15ØØ
```

Nach dem INPUT in Zeile 4Ø brauchen wir einen Test, um festzustellen, ob die Sequenz schon beendet ist. Verwenden wir zum Abschluß "**":

```
42  IF n$ = "**" THEN GO TO 2ØØ
```

um aus der Schleife herauszukommen.

Zeile 1ØØ muß sich ändern, damit der codierte Ton nicht mit BEEP wiedergegeben, sondern gespeichert wird:

```
1ØØ   LET t (q) = s (z) + i
```

Da q die aufeinanderfolgenden Werte 1, 2, 3 etc. erhält, wird der erste Ton in t (1) gespeichert, der nächste in t (2), und so weiter.

Die Schleife endet bei Zeile 11Ø mit:

```
11Ø   NEXT q
```

statt mit dem GO TO von vorher.

Jetzt können wir uns daran machen, von Zeile 2ØØ an die Melodie zu spielen:

```
2ØØ   FOR r = 1 TO q
21Ø   BEEP Ø.5, t (3)
22Ø   NEXT r
```

[Beachte: Diese Schleife geht nur bis q, nicht bis 15ØØ. Grund: Nach dem Verlassen der ersten Schleife enthält q die Zahl der Töne in der Melodie.]

Zur ständigen Wiederholung brauchen wir nur anzufügen:

```
23Ø   GO TO 2ØØ
```

Für den Fall, daß die verschiedenen Veränderungen und Abwandlungen Sie verwirrt haben, finden Sie das endgültige Programm im Abschnitt Fertigprogramme.

Schließlich wollen wir die Musik noch durch ein farbiges Display ergänzen. Mit das Einfachste, was wir tun können, ist, bei jedem gespielten Ton die Randfarbe zu verändern. Nun wäre es schön, wenn wir schreiben könnten:

```
215   BORDER t (r)
```

damit jeder Ton eine eigene Randfarbe hätte, aber das geht nicht, weil der Wertebereich, der sich in t (r) befinden kann, −3 bis 8 ist und der Bereich möglicher Farben nur Ø bis 7. Fügen wir t (r) 3 an, reicht er von Ø bis 11, multiplizieren wir das Ergebnis mit 7/11, so erhalten wir, wie gewünscht, Ø bis 7. Also:

```
215   BORDER INT ((t (r) +3) *7/11)
```

Verändern Sie nun die Paperfarbe jedesmal, wenn ein Ton gespielt wird:

```
216   PAPER 8 * (r/8-INT (r/8)): CLS
```

Das verlangsamt die Dinge ein bißchen, was?

Ersetzen Sie 216 durch:

```
216   PRINT INK ((t (3) +3) *7/11); "□□□";
```

(Es ist nicht unbedingt erforderlich, den Ausdruck mit INT zu bestücken; BASIC tut das von selbst, weil INK keinen Wert besitzen kann, der nicht einer ganzen Zahl entspräche.)

Experimentieren Sie; wahrscheinlich kommen Sie auf Displays, die noch viel gräßlicher aussehen!

16 DEBUGGING III

. . . aber warum den Spectrum nicht dazu bringen, daß er sich die Fehler selbst abgewöhnt?

Beim letzten Beispiel, das wir uns angesehen haben, ging der Debuggingprozeß so vor sich, daß *wir* uns angestrengt haben. Es wäre hübsch, dem Computer, der bis jetzt nur selbstgefällig dagesessen und uns die falschen Antworten gegeben hat, einen Teil der Arbeit aufhalsen zu können.

Die Frage lautet also: Welche Dinge könnte uns der Computer sinnvoll darüber mitteilen, wie er ein Programm ausführt? Hier sind drei Hauptbereiche zu beachten:

1 Welche Werte erhält das Programm in verschiedenen Abschnitten seiner Ausführung für seine Variablen?
2 Wohin geht das Programm (d. h., welche Zeilen werden ausgeführt und in welcher Reihenfolge)?
3 Wie oft geht es dorthin (d. h., wie oft werden bestimmte Zeilen oder Zeilengruppen ausgeführt)?

Manche Computer bieten für die Anzeige zumindest einiger dieser Informationen eine Automatik, aber für ca. DM 500,– (ZX Spectrum mit 16K), kann man (noch) nicht alles haben. Wir müssen deshalb in unsere Programme zusätzliche Zeilen einbauen, um einige dieser Details zu liefern. Sehen wir uns die obigen Punkte der Reihe nach an.

Werte von Variablen anzeigen

Es ist sehr einfach, Variablenwerte überall dort anzuzeigen, wo wir wollen. Wir brauchen nur an einer passenden Stelle im Programm eine PRINT-Anweisung einzufügen. Beispiel: In unserem Mittelwert-Programm könnten wir eine Zeile 55 einfügen:

```
55 PRINT c
```

was uns gestatten würde, die Veränderungen im Wert von c während der Programmausführung zu verfolgen. (Es wäre nicht einmal schwierig, den Computer zu veranlassen, daß er ein Exemplar des Schreibtischtests liefert, den wir von Hand vorgenommen haben – vielleicht versuchen Sie es einmal.)

Das eigentliche Problem hier besteht darin, sparsam und vernünftig auszuwählen, wo Zwischenwerte ausgegeben werden sollen, weil Sie sonst nur endlose Zahlenreihen erhalten, die zu analysieren genauso lange dauert wie ein Schreibtischtest von Hand.

Beim zweiten Lauf des Mittelwert-Programms wurde, wie Sie sich erinnern, durch Zeile 1ØØ eine Fehlermeldung ausgelöst, so daß wir keine Werte

angezeigt bekamen. Ein vernünftiger erster Schritt bestünde darin, eine Zeile 95 einzufügen:

```
95  PRINT s, c
```

Vergessen Sie nicht, daß Zeile 40 abgeändert werden muß:

```
4Ø  IF n < Ø THEN GO TO 95
```

Sonst wird die neue Anweisung nämlich nie ausgeführt. Das hat die Wirkung, daß unser Verdacht sich verstärkt, c enthalte Null und wenig mehr. Fügen Sie nun Zeile 65 ein:

```
65  PRINT c; i; n; s; z
```

so daß wir am Ende jeder Schleife eine vollständige Liste aller Variablen erhalten (der Bequemlichkeit halber in alphabetischer Reihenfolge). Das führt zu einer vereinfachten Form der Tabelle im Schreibtischtest, die aber die wesentlichen Punkte enthält.

Übrigens hier ein flotter Trick: Im Verlauf des Programmtestens wollen Sie vielleicht vorübergehend eine der neuen Zeilen herausnehmen, damit nicht zu viele Variablen auf einmal angezeigt werden. Das heißt, daß man später die ganze Zeile wieder eintippen, oder noch schlimmer, wie bei unserer Zeile 95 auch noch eine zweite Zeile abändern muß, weil Zeile 4Ø sich zurückverwandelt haben wird zu

```
4Ø  IF n < Ø THEN GO TO 1ØØ
```

wenn Zeile 95 herausgenommen wird. Das ist nicht notwendig. Geben Sie am Anfang jeder Zeile, die Sie sperren wollen, einfach ein REM ein. Zeile 95 lautet dann:

```
95  REM PRINT s, c
```

Da das jetzt ein Remark ist (also eine Bemerkung), beachtet der Computer das nicht, aber eine Verzweigung zu Zeile 95 ist durchaus zulässig! Wenn wir die Anweisung wieder brauchen, nehmen wir das REM einfach wieder heraus.

Den Weg verfolgen

Die einfachste Methode, den Weg zu verfolgen, den ein Programm genommen hat, besteht darin, hinter jede Zeile eine PRINT-Anweisung zu setzen, mit der die eben ausgeführte Zeile angezeigt wird. Beispiel: Das Mittelwert-Programm sähe dann so aus:

```
1Ø  LET s = Ø
11  PRINT "1Ø"
2Ø  LET c = Ø
21  PRINT "2Ø"
```

```
30  INPUT n
31  PRINT "30"
    etc.
```

Wieder laufen wir Gefahr, zuviel Information zu produzieren und den Wald vor Bäumen nicht mehr zu erkennen. Wir wollen deshalb bei dieser "Verfolgungs"-Prozedur ein bißchen schärfer auswählen. Die Art von Frage, die ein Ablaufüberwacher gut beantworten kann, lautet: "Verzweigt das Programm dort, wo es das tun soll?" Da es uns darum geht, ist es sinnvoll, unsere Ablaufüberwachung auf Bereiche zu beschränken, wo Verzweigungen stattfinden.

Beispiel: Nehmen wir an, in einem Programm soll ein Monatstag eingegeben werden. Ein solcher Wert läge im Bereich 1–31, also entspräche es vernünftiger Programmpraxis, dafür zu sorgen, daß der Anwender einen solchen Wert eingegeben hat, bevor er fortfährt. Wir könnten einen Code dieser Art schreiben:

```
 50  INPUT t
 60  IF t > 0 OR t < 32 THEN GO TO 200
 70  REM Hier, wenn ungültiger Tag
     .....
200  REM Hier für einen gültigen Tag
     .....
```

Das Programm verhält sich nicht so, wie es sollte, also schieben wir nach den Zeilen 50, 70 und 200 einen Überwachungsbefehl ein:

```
 50  INPUT d
 51  PRINT "* 50 *"
 60  IF t > 0 OR t < 32 THEN GO TO 200
 70  REM Hier, wenn ungültiger Tag
 71  PRINT "* 70 *"
     .....
200  REM Hier für einen gültigen Tag
201  PRINT "* 200 *";
     .....
```

Wir stellen fest, daß, gleichgültig, welchen Eingabewert wir auch nehmen, bei der Ablaufüberwachung stets herauskommt

```
* 50 ** 200 *
```

(Ich verwende Sternchen deshalb, damit die Zeilennummern von Ablaufüberwachern nicht verwechselt werden können mit Zahlen, die das Programm anzeigt. Sie können jedes Sonderzeichen nehmen, das Ihnen gefällt.)

Das Programm kann also nicht dazu gebracht werden, zu Zeile 7Ø zu gehen. Es gibt nur eine vernünftige Schlußfolgerung: Die Bedingung t > Ø OR t < 32 wird jedesmal erfüllt. Von der Logik her ist das so – jede Zahl ist *entweder* größer als Ø *oder* kleiner als 32. Wir hätten schreiben sollen:

```
6Ø  IF t > Ø AND t < 32 THEN GO TO 2ØØ
```

Die Verwechslung von AND mit OR ist ein häufig auftretender Fehler, und zwar deshalb, weil wir diese Wörter im normalen Sprachgebrauch weniger exakt verwenden. Im vorliegenden Fall müssen *beide* Bedingungen erfüllt sein, bevor wir zu einem gültigen Tag kommen, also ist die Verbindung AND erforderlich.

Programmprofile

Ein Programmprofil zeigt an, wie oft jede Zeile eines Programms ausgeführt worden ist. Wie gewohnt, wird hier gern maßlos übertrieben, und wir sollten uns mit Überlegung aussuchen, bei welchen Programmteilen wir ein Profil anstreben. Das geht ganz leicht. Angenommen, wir wollen feststellen, wie oft Zeile 42Ø eines bestimmten Programms ausgeführt wird. Wir setzen zu Beginn eines Programms eine Zählung auf Null und erhöhen jedesmal, wenn Zeile 42Ø durchlaufen wird, um 1:

```
   5  LET zc = Ø
      .....
 42Ø  LET a = a * (z - 1)
 421  LET zc = zc + 1
      .....
 8Ø9  PRINT zc
 81Ø  STOP
```

Sehen wir uns ein konkretes Beispiel an. Das folgende Programm soll maximal 20 Werte annehmen, wobei mit einer Null abgeschlossen wird, und sie in aufsteigender Reihenfolge ordnen. Wenn die Eingabefolge also lautet:

```
3
8
1
4
2
Ø
```

sollte herauskommen

```
1
2
3
4
8
```

Die Null sollte nicht erscheinen, weil sie nur ein Begrenzer ist.

```
 1Ø  DIM a (2Ø)
 2Ø  FOR z = 1 TO 2Ø
 3Ø  INPUT a (b)
 4Ø  IF a (z) = Ø THEN GO TO 6Ø
 5Ø  NEXT z
 6Ø  LET n = z
 7Ø  FOR z = 1 TO n
 8Ø  IF a (z) < a (z + 1) THEN GO TO 13Ø
 9Ø  LET t = a (z)
1ØØ  LET a (z) = a (z + 1)
11Ø  LET a (z + 1) = t
12Ø  LET f = 1
13Ø  NEXT z
14Ø  IF f = 1 THEN GO TO 65
15Ø  FOR z = 1 TO n
16Ø  PRINT a (z)
17Ø  NEXT z
```

Das Programm leistet nicht ganz, was es soll. (Tippen Sie es ein und probieren Sie es aus.) Es geht vielmehr in eine Endlos-Schleife.

Wo fangen wir mit dem Suchen also an? Die erste Schleife (2Ø–5Ø) sieht ganz harmlos aus, die letzte (15Ø–17Ø) zeigt lediglich den Inhalt des Arrays a an. Deshalb spricht die Vernunft dafür, sich mit der Schleife von 7Ø bis 13Ø zu befassen. Aus Zeile 8Ø ergibt sich, daß manchmal alle Anweisungen in der Schleife ausgeführt und manchmal die von 9Ø bis 12Ø nicht beachtet werden. Wir brauchen also zwei Profilzähler c1 und c2, die zählen, wie oft in die Schleife hineingegangen und wie oft der letzte Teil der Schleife ausgeführt wird.

Das können wir erreichen mit:

```
 67  LET c1 = Ø
 68  LET c2 = Ø
 75  LET c1 = c1 + 1
125  LET c2 = c2 + 1
132  PRINT c1, c2
```

Wenn wir schon dabei sind, können wir auch gleich den Inhalt des Arrays am Ende jeder Schleife anzeigen lassen, weil offenkundig ist, daß darin Zahlen umgeschaufelt und kaum andere Variablen verwendet werden. Also:

```
134  FOR q = 1 TO n          [weil 1 TO n der relevante Teil des
135  PRINT a (q);            Arrays zu sein scheint]
136  NEXT q
137  PRINT
```

Probieren wir Datenmengen aus und warten wir ab, was sich tut. Wenn wir eingeben

```
3
6
1
8
5
Ø
```

erhalten wir:

```
6  4
3165ØØ
6  4
13ØØ56
6  2
1ØØ356
6  2
ØØ1356
6  2
```

```
ØØ1356
6  1
ØØ1356
6  1
ØØ1356
```

und so weiter, bis der Speicherplatz ausgeht.

Tja. Die Werte scheinen zwar in eine Reihenfolge zu kommen, aber wir haben die "8" verloren, und woher kommen die Nullenpaare? Außerdem geht das Programm 6mal beharrlich durch die Hauptschleife, aber die Läufe durch die Unterschleife werden immer seltener, bis sie bei 1 sind, wo sie bleiben.

Eine der Nullen ist offenkundig der Begrenzer, die andere ein Element des Arrays, das nicht während des Durchlaufs gesetzt, sondern durch das System auf Null aktualisiert wird. Mit anderen Worten: Das Programm hat es mit zwei Werten zuviel zu tun. Schreiben wir Zeile 6Ø also um:

```
6Ø   LET n = z - 2
```

und versuchen wir es noch einmal. Die Hoffnung währet ewiglich . . .
Wir erhalten

```
4          2
3165
4          2
1356
4          Ø
1356
1
3
5
6
```

Immerhin ein Fortschritt – die Nullen sind wir losgeworden. Aber unsere 8 haben wir immer noch nicht wieder.

Schwer zu erkennen, wo wir sie verloren haben. Vielleicht ist sie noch da und wird nur nicht angezeigt. Wo zeigen wir sie mit PRINT an? In den Zeilen 15Ø–17Ø. Der Bereich 1 TO n muß zu klein sein. Erhöhen wir ihn um 1:

```
15Ø   FOR z = 1 TO n + 1
```

Und weil wir schon dabei sind: Der Ablaufüberwacher in Zeile 134 hat vermutlich mit denselben Problemen zu kämpfen. Das wollen wir auch gleich beheben:

```
134   FOR q = 1 TO n + 1
```

Je nun; von neuem in die Bresche, Freunde, laßt es uns einmal noch versuchen . . .

Diesmal erhalten wir (für dieselben Daten):

4 2

31658

4 2

13568

4 ∅

13568

1

3

5

6

8

Heureka! Wir haben die Nuß geknackt. Alles läuft ideal. Wirklich? Probieren wir:

3

5

2

1

5

∅

Jetzt erhalten wir:

4 3

32155

4 3

21355

4 2

12355

4 1

12355

4 1

12355

4 1

12355

etc.

Die richtige Lösung wird zwar erreicht, aber das Programm kommt nicht mehr aus der Schleife heraus. Uns fällt auf, daß in diesem Fall c2 nie auf Null heruntergeht. Man wird also davon ausgehen dürfen, daß genau dies das Programm beendet.

Was entscheidet darüber, ob das Programm in die Subschleife eintritt oder nicht? Zeile 8Ø:

```
8Ø  IF a (z) < a (z + 1) THEN GO TO 13Ø
```

Der Unterschied zwischen den beiden Datenmengen ist der, daß die zweite zwei gleiche Werte enthält. Da 5 nicht weniger als 5 ist, wird jedesmal dann, wenn die doppelte 5 erscheint, die Subschleife ausgeführt. Vielleicht sollte die Frage lauten:

```
8Ø  IF a (z) < = a (z + 1) THEN GO TO 13Ø
```

Diesmal funktioniert alles.

4 2

32155

4 2

21355

4 1

12355

4 Ø

12355

1

2

3

5

5

Jetzt läuft es wie geschmiert, und wir können die Testzeilen herausnehmen.

Ich hoffe, ich habe hier zwei wichtige Punkte veranschaulichen können. Erstens: Wir mußten nicht wissen, wie die Prozedur genau funktioniert. Wenn Sie das gründlich durchgearbeitet haben, wird es inzwischen ziemlich klar sein, und ein paar Schreibtischtests würden Sie vermutlich davon überzeugen, daß

Sie es begriffen haben. (Schreibtischtests sind ein guter Weg, wenn man erkennen möchte, wie Computerprozeduren ablaufen. Ich habe bei einem unklaren Codeteil – der selbstverständlich von einer anderen Person stammte – oft rund ein Dutzend Tests angestellt, bevor mir wirklich ganz klar wurde, was vorging.) Zweitens: Man neigt immer gern zu der Ansicht, die Arbeit sei getan, wenn ein Programm zum erstenmal erfolgreich läuft, und man dürfe sich in der Eckkneipe jetzt ein Glas Bier vergönnen. Wie wir sehen konnten, ist die Arbeit eben *nicht* getan, weil es andere Datenmengen geben kann, an denen das Programm scheitert. Außerdem hat die Kneipe schon vor eineinhalb Stunden zugemacht, wenn Ihnen die Zeit beim Schreiben von Code genauso schnell verfliegt wie mir.

17 STRINGS

Computer sind nicht bloß Zahlenknacker.
Sie können auch Zeichen knacken, das heißt, mit Symbolen umgehen.

Der Postbote erscheint . . . und bringt einen Brief für Sie. Einen sehr persönlichen Brief. "Lieber Herr Frempelmotz", steht da, "Sie sind aus einer kleinen Gruppe von Personen in Unterschwer-Steinhausen ausgewählt worden und erhalten völlig kostenlos ein erstklassiges Paar Gummistiefel mit Betonfutter . . ." Sehr erfreulich. Aber die alte Frau Kniefschatten nebenan hat fast genau den gleichen Brief erhalten. Um genau zu sein: Ganz Unterschwer-Steinhausen hat ebenso wie die Einwohnerschaft mehrerer Kantone solche Briefe bekommen.

Das geht so:

```
 1Ø  INPUT "Wie heißen Sie?"; n$
 2Ø  INPUT "In welcher Stadt leben Sie?"; t$
 3Ø  INPUT "In welcher Stadt wohnen Sie?"; s$
 4Ø  INPUT "Welche Hausnummer haben Sie?"; h
 5Ø  PRINT n$
 6Ø  PRINT h; ",□"; s$
 7Ø  PRINT t$
 8Ø  PRINT "Lieber □"; n$; ","
 9Ø  PRINT "□□□ Sie sind ausgesucht worden aus einer"
1ØØ  PRINT "kleinen Gruppe wohnhaft in"
11Ø  PRINT t$; "□ und erhalten"
12Ø  PRINT "voellig kostenlos * ein"
13Ø  PRINT "herrliches Paar Gummistiefel"
14Ø  PRINT "mit Betonfutter. Wir sind"
15Ø  PRINT "sicher, □"; n$; "□ daß Sie"
16Ø  PRINT "dieses großzuegige Angebot"
17Ø  PRINT "nuetzen wollen und daß die"
18Ø  PRINT "anderen Einwohner von"
19Ø  PRINT t$; "□ vor Neid gelb wie die"
2ØØ  PRINT "Papageien werden duerften."
```

```
21Ø  PRINT "□□□ Mit unverbindlichem Gruß."
22Ø  PRINT "□□□□□ Franz X. Protzenfuß"
23Ø  PRINT "□□□□□ Schmonzettenhandel."
24Ø  PRINT,, "* Postspesen DM 1043.22 extra."
```

Drücken Sie RUN und geben Sie mit INPUT (z. B.) ein "Lisa Hurtig"; "Schmerdorf"; "Knoedelsteig"; "666". Versuchen Sie es mit anderen Namen und Adressen. Hmmmm . . .

Stellen Sie sich das Ganze nun vor, wie das aus einer Datenbank Namen und Adressen zugeführt bekommt und in der Stunde Tausende von Briefen ausspuckt.

Sieht man einmal davon ab, wie himmelschreiend der ganze Unfug ist, das Interessante daran ist eben, daß überhaupt keine *Computerberechnungen* erforderlich sind. Es geht nur um Speicherplatz und die ganz simple Handhabung von geschriebenem Text. Der Computer bewältigt das, weil er so gut wie Zahlen auch *Ketten* speichern kann, die im Computerjargon *Strings* heißen. Das ist es, was die Dollarzeichen "$" bezeichnen, obschon im gegebenen Zusammenhang vielleicht auch Vater Freud ein Wörtchen mitzureden hätte.

Ein *String* (bei diesem Ausdruck wollen wir bleiben) ist eine Folge von *Zeichen*. Die Zeichen sind im Anhang A des Handbuchs oder auf dem Datenblatt aufgeführt. Jedes besitzt einen CODE, mit dem wir uns dann gleich befassen wollen.

Hier ist ein String:

stringstringstringstringstring.

Hier ein zweiter

ab 334 * . / > > > > < < − b + + + + + qqj.

(Wenn Sie die Punkte eintippen, sind auch sie Bestandteil des Strings!)

Ein String kann nur ein Zeichen lang sein, etwa <, oder *gar* keine Zeichen lang!

Um ein Stringvariable zuzuteilen, müssen Sie den String in Anführungszeichen setzen:

```
1Ø  LET a$ = "stringstringstringstringstring"
```

Jede Stringvariable muß ein einzelner Buchstabe sein, gefolgt von dem $-Zeichen. Bei einem String von keinem Zeichen Länge genügt LET a$ = "".

Strings tun alles, was man von ihnen verlangt, wenn man mit ihnen umgehen kann. Nehmen Sie einen Einzelzeichenstring wie

3

Man kann ihn sehen als

a   eine Zahl, 3
b   einen String, "3"

und Sie können auf verschiedene Art und Weise vom einen zum anderen umsteigen. Angenommen, wir sehen die 3 wirklich als String und teilen sie zu:

```
10   LET a$ = "3"
```

Gehen wir davon aus, Sie wollen 3 + 5 berechnen. Es hat keinen Sinn, vom Spectrum zu verlangen:

```
20   LET b = a$ + 5
30   PRINT b
```

Das funktioniert nicht – probieren Sie es aus, wenn Sie mir nicht glauben. Und warum nicht? Dem dummen Ding ist mitgeteilt worden, die 3 sei als *String* zu betrachten, und es weiß nicht, daß das auch eine *Zahl* ist. Aber – aha! – wir können ihn mit Hilfe von VAL in eine Zahl *umwandeln*. Probieren Sie

```
20   LET b = VAL a$ + 5
30   PRINT b
```

Wenn der Computer einen String erhält, der auch ein arithmetischer Ausdruck ist wie

```
10   LET a$ = "2 + 2 + 5 * 3"
```

weiß er im allgemeinen nicht, wie das als Zahl zu berechnen wäre; er sieht das nur als eine Folge von *Zeichen*

2 + 2 + 5 * 3

Beispielsweise können Sie das sechste Zeichen herausnehmen:

```
20   PRINT a$ (6)
```

und erhalten * als Antwort. (Das kann nützlich sein: Als arithmetischer Ausdruck entspricht es 19, das kein sechstes Zeichen *besitzt*.) Wenn Sie es aber in eine Zahl verwandeln wollen, wird

```
20   PRINT VAL a$
```

als Antwort 19 bringen.

Eine Zahl können Sie auf zweierlei Weise in einen String verwandeln. Erstens, indem Sie sie in Anführungszeichen setzen, also "3336" oder eine beliebige Zahl. ABER: Wenn Sie innerhalb eines Programms sind und a + b oder was auch immer berechnen, was 3336 *ergibt*, hat es natürlich keinen Zweck, "a + b" zu schreiben, in der Hoffnung, dabei käme "3336" heraus; vielmehr erhalten Sie einen String aus drei Zeichen

a + b

und das ist etwas völlig anderes.

CHR$ verwandelt eine Zahl zwischen Ø und 255 entsprechend der Codeliste im Handbuch, Anhang A (oder Datenblatt) in ein Einzelzeichen. Beispiel: CHR$ 96 ist das Pfundzeichen £. Manche Zahlen werden nicht verwendet.

CODE wirkt umgekehrt: CODE "£" ist 96. Wenn Sie CODE "£335/h" versuchen, erhalten Sie trotzdem 96; der Computer sieht nur das erste Zeichen.

LEN sagt Ihnen, wie lang der String ist.

Das weitaus Interessanteste an Strings ist, daß man Teile herausnehmen kann, die *Substrings* heißen. Die Anweisung

```
a$ (3 TO 7)
```

holt den String heraus, welcher der Reihe nach aus dem 3., 4., 5., 6. und 7. Zeichen von a$ besteht. Sie können anstelle von 3 und 7 jede andere Zahl verwenden. Und a$ (5) holt nur das 5. Zeichen, in diesem Fall "n".

Beispiel: Sie wollen eine Zahl zwischen 1 und 7 eingeben und bestimmen, welcher Wochentag das ist (mit 1 als Sonntag, 2 als Montag, etc.). Eine umständliche Methode wäre diese:

```
1Ø   INPUT n
2Ø   IF n = "1" THEN PRINT "Son"
3Ø   IF n = "2" THEN PRINT "Mon"
     .....
```

bis alle sieben Tage durch sind.

Aber mit Substrings

```
1Ø   LET a$ = "SonMonDieMitDonFreSam"
2Ø   INPUT n
3Ø   PRINT a$ = (3 * n – 2 TO 3 * n)
```

ist dasselbe mit fünf Zeilen weniger zu erreichen; siehe dazu TAGEFINDER, Seite 162.

Sie können Strings mit + auf folgende Weise aneinanderhängen:

```
"hot" + "dog" = "hotdog"
```

oder sie mit < "alphabetisch" ordnen. Schlagen Sie im Handbuch nach und experimentieren Sie. Überlegter Einsatz von Strings kann oft viel Speicherplatz sparen.

Das folgende Programm bestimmt die Anfangsbuchstaben eines Namens.

```
1Ø   INPUT "Wie ist dein voller Name?"; n$
2Ø   LET n$ = "□" + n$
3Ø   FOR i = 1 TO LEN n$
4Ø   IF n$ (i) = "□" AND i < LEN n$ THEN PRINT n$ (i + 1); ".";
5Ø   NEXT i
```

Drücken Sie RUN und geben Sie Ihren eigenen Namen ein. Geben Sie "Ausgemachte Eleganz Gesellschaft" ein und sehen Sie sich an, ob "A. E. G." herauskommt.

So, wie das Programm dasteht, ist es nicht perfekt; wenn Sie zwischen Wörter zusätzliche Zwischenräume setzen, sieht die Ausgabe recht unschön aus. Man braucht vorne nur eine zusätzliche Leerstelle einzuschalten, damit das Leben einfacher wird, und dann davon auszugehen, daß jedes Zeichen *nach* einer Leerstelle ein Anfangsbuchstabe ist. Verwenden Sie Punkte, damit es sauberer aussieht.

Aufgabe

Ändern Sie das Programm so ab, daß es wiederholte Leerstellen nicht beachtet. Ein Weg besteht darin, alles abzusuchen und sämtliche Leerstellen, die einer ersten folgen, zu löschen. Damit das j-te Zeichen aus einem String n$ gelöscht wird, schreiben Sie

```
LET n$ = n$ (TO j − 1) + n$ (j + 1 TO)
```

Wenn Sie die Zahlen vor oder nach dem "TO" weglassen, geht der Computer davon aus, daß sie Anfang und Ende sind.

18 DATA

Es gibt ökonomischere Methoden, viele Variablen zu bestimmen, als durch die Verwendung von LET-Anweisungen: DATA.

Über der Taste D steht in grüner Schrift das Wort "DATA". Mit diesem Befehl können Sie vermeiden, lange Folge von Anweisungen in der Art "LET a (37) = 242", "LET a (38) = 243", . . . etc. eingeben zu müssen. Beispiel (nur, damit Sie sich mit dem Befehl vertraut machen können):

```
10  DATA 1, 2, 3, 4, 5, 6, 7
20  FOR i = 1 TO 7
30  READ x
40  PRINT x
50  NEXT i
```

Sie sollten als Ausgabe wieder die Zahlen 1–7 erhalten. Der READ-Befehl heißt im Grunde "LET x = die nächste Zahl auf der DATA-Liste". Jedesmal, wenn eine READ-Anweisung auftaucht, sucht der Computer die DATA-Zeilen ab, findet den letzten Posten, den er gelesen hat, und gibt den nächsten als den Wert der Variablen nach der READ-Anweisung ein. Beim ersten Durchgang liest er also x als 1, beim nächstenmal als 2, und so weiter.

Eine nützliche Eigenschaft von DATA ist, daß der Computer alle DATA-Zeilen in einem Programm als eine einzige aneinandergekettete Liste betrachtet. Sie können diese Zeilen hinstellen, wo Sie wollen (allerdings geht es oft schneller, wenn das irgendwo am Anfang geschieht, weil der Computer sie ja absucht). Versuchen Sie das Programm umzuschreiben, etwa so:

```
10  FOR i = 1 TO 7
20  READ x
30  DATA 1, 2, 3, 4
40  PRINT x
50  DATA 5
60  NEXT i
70  DATA 6, 7
```

Das funktioniert genauso – selbst wenn die DATA-Zeile sich innerhalb einer FOR/NEXT-Schleife befindet. (DATA ist eine Ausnahme von der Regel, daß Zeilen in numerischer Reihenfolge ausgeführt werden.)

Wenn das schon alles wäre, was Sie damit anfangen können, wäre das nicht sehr eindrucksvoll. Hier ist aber noch eine typischere Anwendungsform:

```
10  DATA 100, 50, 150, 100, 100, 150, 50, 100, 100, 50
20  READ x, y
30  PLOT x, y
40  FOR i = 1 TO 4
50  LET x0 = x : LET y0 = y
60  READ x, y
70  DRAW x – x0, y – y0
80  NEXT i
```

Damit müßte eine Rautenform gezeichnet werden: Die DATA-Liste liefert die Koordinaten der Ecken, wobei die untere Ecke für das Ende ebenso wie für den Anfang wiederholt wird.

Daraus wollen wir etwas ganz Heißes machen . . .

```
10  DATA 2, 0, 6, 0, 9, 1, 15, 0, 16, 1, 16, 12, 15, 13, 14, 12,
    14, 2, 10, 2, 11, 6, 8, 12, 10, 15, 9, 18, 8, 22, 7, 18, 3, 18, 2,
    22, 1, 18, 0, 15, 2, 12, 0, 6, 0, 3, 2, 0
20  READ x, y
30  PLOT x, y
40  FOR i = 1 TO 23
50  LET x0 = x : LET y0 = y
60  READ x, y
70  DRAW x – x0, y – y0
80  NEXT i
```

Wenn Sie analysieren, was hier geschieht, werden Sie feststellen, daß zuerst der Punkt mit den Koordinaten 2, 0 gezeichnet und er dem Punkt 6, 0 angefügt wird, dann 9, 1, und so weiter. Die Zahlen folgen in der DATA-Liste aufeinander.

Damit das Bild mehr in die Schirmmitte rückt und nicht so zusammengequetscht aussieht, verändern Sie Zeile 30 zu

```
30  PLOT 100 + x, 60 + y
```

Die DATA-Liste selbst wurde anhand einer Rohskizze auf Millimeterpapier erarbeitet, die Koordinaten wurden von dort abgelesen. Das ist wie bei den Bildern, die aus vielen Punkten bestehen, die man miteinander zu verbinden hat. Nur haben wir die Orte der Punkte hier mit Koordinaten anzugeben.

Für eine einzige Katze ist das aber immer noch viel Arbeit. Um dicke Katzen, dünne Katzen und sogar auf den Kopf gestellte oder seitenverkehrte

Katzen zu erhalten, verwandeln Sie einfach die Daten. Ändern Sie Zeile 30 wie oben ab, um genug Platz zu schaffen; fügen Sie diese Zeile an:

```
5   INPUT a, b
```

und verändern Sie Zeile 70 zu

```
7Ø   DRAW a * (x - xØ), b * (y - yØ)
```

Nach RUN müssen Sie nun zwei Zahlen eingeben. Seien Sie anfangs noch nicht zu ehrgeizig. Versuchen Sie es mit a = 1, b = 2, und a = 2, b = 1. Probieren Sie dann a = 2, b = – 1; a = – 2, b = – 1. Wenn Sie die gesehen haben, nehmen Sie, was Sie wollen! Aber Vorsicht: Das Programm ist nicht dagegen gesichert, vom Bildschirm zu rutschen, und Bruchwerte von a und b bewirken durch Rundungsfehler seltsame Erscheinungen. Einen Kniff, mit dem sich dieses Problem umgehen läßt, finden Sie in SPIRALEN, Seite 168.

Jetzt wollen wir eine ganze Reihe von Katzen zeichnen. Dazu können wir eine Schleife verwenden, aber wir brauchen eine Methode, die READ-Anweisung zum Anfang der Data-Liste zu versetzen. Das geht mit RESTORE. Wir fügen also die Zeilen an

```
 15   FOR t = 1 TO 8
 9Ø   RESTORE
1ØØ   NEXT t
```

und verändern Zeile 3Ø zu

```
3Ø   PLOT 5Ø + x + 2Ø * t, 5Ø + y
```

Das ist alles.

Aufgaben

1 Setzen Sie die Katzen auf dem Bildschirm höher, während sie nach rechts wandern, als säßen sie auf einer Treppe.
2 Geben Sie mehr DATA ein, um die Treppe zu zeichnen.
3 Verändern Sie die DATA-Liste nach einem Atlas, so daß das Programm eine Karte von Australien zeichnet. Stellen Sie fest, wie Australien auf den Kopf gestellt aussieht. (Die Australier glauben ohnehin, das sei der Fall.)
4 Wenn Sie einen halben Nachmittag Zeit haben, stellen Sie DATA für eine Weltkarte zusammen.

19 DEBUGGING IV

Läuft das Programm wirklich?

Wie können wir schlüssig beweisen, daß ein Programm genau das leistet, wozu es geschrieben worden ist? Ich will mich nicht auf komplizierte philosophische Erörterungen einlassen (wir wären auf dem besten Weg dazu), aber grob gesprochen ist das so, als wolle man von einem Astronomen wissen, ob morgen die Sonne aufgehen wird. Wenn er sehr pedantisch ist, gibt er vielleicht zur Antwort, die Erde kreise jetzt schon sehr lange um die Sonne. Wir besäßen ein System physikalischer Gesetze, das darauf hindeute, sie werde das auch künftig tun. Man dürfe also einiges darauf verwetten, daß das auch morgen noch der Fall sein werde. Er würde aber hinzufügen, er könne nicht mit Bestimmtheit wissen, ob unsere physikalischen Gesetze richtig seien. Das, was wir seit Jahrtausenden beobachteten, könne auch die Manifestation eines viel komplexeren Gesetzes sein, dessen Wirkung morgen darin bestehen möge, die Erdrotation umzukehren oder den Planeten ganz aus seiner Bahn zu führen.

Entsprechend: Nur, weil ein Programm bei der Eingabe der ersten tausend Datenmengen richtig läuft, besteht noch keine absolute Gewähr dafür, daß das beim tausendundersten Mal auch der Fall sein wird. Tatsächlich tauchen Fehler oft erst Monate oder sogar Jahre nach der scheinbar erfolgreichen Fertigstellung eines Programms auf, nachdem es zwanzig-, dreißigmal oder sogar hundertfach ohne Probleme gelaufen ist. Eigentlich kein Wunder. Schließlich wird der Programmierer gerade jene Bedingungen am ehesten übersehen, die am seltensten vorkommen.

Hier ein Beispiel:

Wir schreiben ein Programm für das E-Werk Hinterniederhofen zur Bearbeitung der Kundenkonten. Man teilt uns mit, daß es zwei Tarife, A und B, gibt. Bei Tarif A zahlt der Kunde im Vierteljahr eine Grundgebühr von 15 Mark und die verbrauchten Strommengen mit 40 Pfennig pro Einheit. Bei Tarif B hat der Kunde keine Grundgebühr zu entrichten und zahlt 7Ø Pfennig je Einheit. Wir verfassen einen Code folgender Art:

```
1ØØ  INPUT t$
1Ø5  INPUT Einheiten
11Ø  IF t$ = "a" THEN GO TO 3ØØ
12Ø  IF t$ = "b" THEN GO TO 14Ø
13Ø  GO TO 5ØØØ
14Ø  LET Rechnung = 70 * Einheiten/1ØØ
15Ø  PRINT Rechnung
16Ø  GO TO 1ØØ
```

```
 3ØØ  LET Rechnung = 15 + 4Ø * Einheiten/100
 31Ø  PRINT Rechnung
 32Ø  GO TO 1ØØ
      .....
5ØØØ  PRINT "ungueltiger Tarif"
5Ø1Ø  STOP
```

Fein. Ich weiß, der Code könnte überlegter aufgebaut sein, und in Wirklichkeit würden wir mehr Information brauchen, etwa Namen und Kontonummer des Kunden, aber Sie sehen, was gemeint ist.

Wir testen also diesen Codeteil, er läuft wunderschön, und wir entfernen uns, in den Bart murmelnd, es sei reine Vergeudung unserer erstaunlichen Talente, derartige Schlafmützenprogramme schreiben zu müssen.

Und das läuft prima, jahrelang, bis eines Tages eine Rechnung über Ø.ØØ Mark ausgestellt wird. Natürlich fällt das keinem auf, weil Tausende von Rechnungen anfallen und vermutlich ohnehin automatisch kuvertiert werden. Der Empfänger ist verwundert und amüsiert sich vermutlich, weil die Rechnung wieder einmal beweist, wie blöd Computer sind. Es scheint aber keinen Sinn zu haben, etwas zu unternehmen. Er wirft das Ding in den Papierkorb. Leider haben wir im gleichen Zusammenhang ein zweites Programm zur Speicherung der Absendedaten aller Rechnungen geschrieben. Wenn keine Bestätigung eingeht, daß die Rechnung innerhalb eines Monats bezahlt worden ist, erfolgt eine letzte Mahnung. Diesmal ist der Empfänger eher verärgert als belustigt, wirft aber auch die Mahnung weg. Von da an geht's bergab. Das Programm für die Prüfung des Zeitraums zwischen Rechnungsstellung und Geldeingang schickt eine Anweisung an die technische Abteilung, dem Kunden den Strom abzuschalten, falls er nach 6Ø Tagen immer noch nicht bezahlt hat.

Was war passiert? Ganz einfach! Der Kunde ist Rentner. Er hat ein Angebot für einen Winter-Pauschalaufenthalt genutzt, wie Reisebüro es älteren Mitbürgern anbieten. Er war knapp über drei Monate im Ausland und hat in einem vollen Rechnungszeitraum keinen Strom verbraucht. Außerdem ist er Energiesparer und zahlt deshalb nach Tarif B. Aus diesem Grund hat das System eine Forderung über Null Mark ausgedruckt. Freilich passiert so etwas nicht oft, weil nur sehr wenige Menschen so lange von zu Hause weg sind; außerdem dürften Abnehmer des Tarif B ziemlich dünn gesät sein. Damit das Problem überhaupt auftreten kann, muß der Kunde beide Bedingungen erfüllen.

Sobald der Fehler einmal erkannt ist, läßt er sich ganz leicht abstellen:

```
145  IF Rechnung = Ø THEN GO TO 1ØØ
```

so daß der Druckauftrag vermieden wird. Das geschilderte Problem soll bei einem frühen Computersystem aufgetreten sein. Ich weiß nicht, ob es nicht einfach erfunden ist, finde aber, daß es auf alle Fälle in hübscher Weise veranschaulicht, wie ein Fehler praktisch ewig verborgen bleiben kann.

Die Moral: Wenn Sie Daten erfinden, um ein Programm zu testen, tun Sie das nicht willkürlich. Wählen Sie Werte an und nahe bei Verzweigungswerten im Programm. Wenn es in einer Zeile heißt

```
3Ø5  IF u < 3Ø THEN GO TO 5ØØ
```

dann lassen Sie einen Test bei 29.999 und einen zweiten bei u = 3Ø.ØØØ1 laufen. Vielleicht haben Sie nämlich gemeint:

```
3Ø5   IF u < = 30 THEN GO TO 5ØØ
```

Wenn Sie nur bei u = 15 und u = 16Ø testen, fällt Ihnen der Fehler nicht auf.

Achten Sie darauf, Testdaten so auszusuchen, daß irgendwann jeder Programmteil getestet wird. Und auf jeden Fall sollten Sie genau wissen, wie die Lösung für jeden Satz Testdaten ausfallen muß.

20 KURVEN GRAFISCH DARSTELLEN

Nur für mathematisch Interessierte!

Von einer *graphischen* Darstellung mit Hilfe von Koordinaten werden Sie schon gehört haben. Wir wollen trotzdem kurz rekapitulieren. Man beginnt mit zwei Geraden, der x-Achse und der y-Achse, die im rechten Winkel zueinander stehen. Wir können daran Entfernungen x und y markieren (negative Zahlen werden auf der x-Achse nach links, auf der y-Achse nach unten angezeichnet) und mit den beiden Entfernungen einen Punkt bestimmen, der die *Koordinaten* x und y hat. Das ist ganz wie bei den Pixels (Abbildung 20.1.)

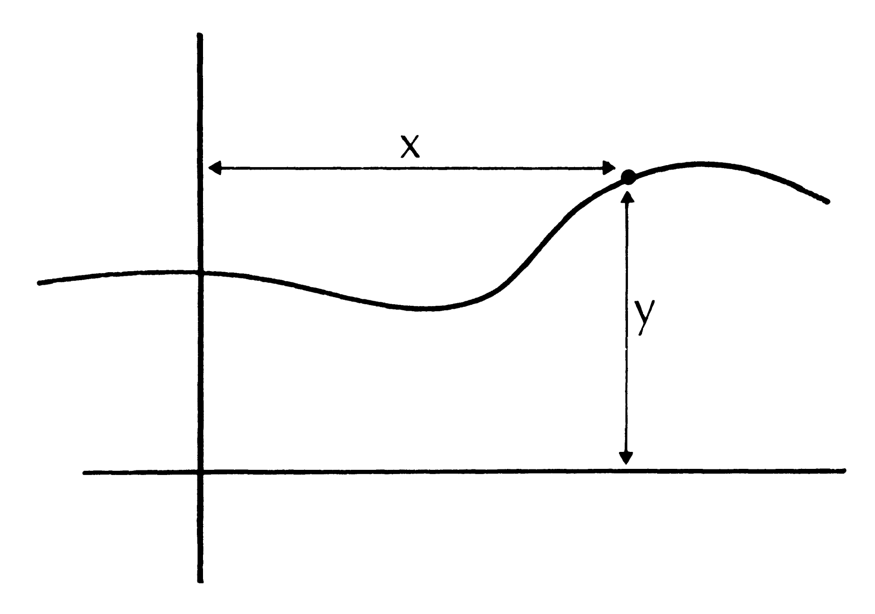


Abbildung 20.1 Koordinaten für Kurvendarstellung

Wenn wir uns jetzt vorstellen, daß x entlang der x-Achse weiterwandert und der Wert von y sich auf eine Art und Weise verändert, die von x abhängt, wird sich auch der Punkt mit den Koordinaten (x, y) bewegen; in der Regel beschreibt er eine *Kurve*. Wenn Sie eine Formel dafür nennen, wie y von x abhängt, meinetwegen $y = x - 3$, haben Sie die Formel für diese Kurve gegeben. (Mit dieser Idee, 1637 von Descartes entwickelt, können Sie geometrische Kurven algebraisch darstellen. Das ist übrigens der Ausgangspunkt für höhere Mathematik.)

Mit PLOT-Anweisungen können wir ähnliche Kurven darstellen:

```
1Ø   FOR j = Ø TO 255
2Ø   PLOT j, j/2
3Ø   NEXT j
```

Sie müßten eine Reihe von Pixels erhalten, die auf dem Bildschirm von unten links nach oben rechts klettern. Ein Mathematiker würde vom Graph der Funktion y = x/2 sprechen.

Sie können das abändern und eine Vielzahl grafischer Darstellungen erreichen. Dazu brauchen Sie nur j/2 durch einen anderen Ausdruck mit j zu ersetzen. Beispiel: Um die Quadratwurzel von j als eine Funktion von j darzustellen, brauchen Sie aus j/2 nur SQR j zu machen und erhalten:

```
1Ø   FOR j = Ø TO 255
2Ø   PLOT j, SQR j
3Ø   NEXT j
```

Schon ein wenig interessanter, nicht?

Experimentieren Sie. Verändern Sie j/2 immer wieder. Probieren Sie das Folgende aus:

```
(Parabel)          2Ø PLOT j, .ØØ2 * j * j
(Sinuskurve)       2Ø PLOT j, 8Ø * SIN (j/20) + 8Ø
(Kosinuskurve)     2Ø PLOT j, 8Ø * COS (j/2Ø) + 8Ø
(Kettenlinie)      2Ø PLOT j, EXP (.Ø2 * (j – 12Ø))
                       + EXP (.Ø2 * (12Ø – j)) * 1Ø
```

. . . Augenblick mal, wieso sind die so kompliziert? Was *macht* er denn da?

Es gibt Probleme, wenn Sie in j als Ausdruck einfach verwenden, was Ihnen gerade so einfällt. Die Frage ist nämlich, was auf den Bildschirm paßt. Der Wert der vertikalen Koordinate muß zwischen 0 und 175 liegen, sonst wird der Spectrum überfordert und sperrt den Laden zu. (Das arme Ding kann nichts anderes darstellen und ist sauer, wenn man es dazu auffordert.) Sie müssen also *normieren*, wie das in der Fachsprache heißt, also den Maßstab verkleinern, damit alles hineinpaßt. Sie werden gleich sehen, warum, wenn Sie das Nachstehende ausprobieren

```
2Ø   PLOT j, j * j
2Ø   PLOT j, SIN j
2Ø   PLOT j, COS j
2Ø   PLOT j, EXP (j) * EXP (– j)
```

Dieses Problem läßt sich natürlich umgehen, – siehe weiter unten bei NORMIEREN. Aber bevor Sie dort nachsehen, hier noch ein paar interessante Funktio-

nen, die wirklich hineinpassen, weil ich sie mit Bedacht danach ausgesucht habe.

```
20  PLOT j, EXP (– j/80) * SIN (j/8) * 80 + 80
20  PLOT j, 80 + 80 * COS SQR (2 * j)
20  PLOT j, ABS (j – 127)
20  PLOT, j, 80 + 60 * LN (1 + ABS SIN (j * .125))
20  PLOT j, 12 * ABS (j/4 – 30) ↑ .666
20  PLOT j, 40 * ABS (j/4 – 30) ↑ .25
20  PLOT j, 160 * EXP (– .1 * (j/4 – 30) * (j/4 – 30))
```

Damit Sie keine komplizierten Ausdrücke schreiben müssen, können Sie die Formel stufenweise aufbauen, etwa so:

```
20  LET t = j/24
25  PLOT j, 120 + .1 * t * (t – 2) * (t – 4) * (t – 6) * (t – 8) * (t – 10)
```

Normieren

Zunächst wollen wir nur mit Funktionen für *positive* Zahlen arbeiten und *positive* Werte nehmen: Gut geeignet dafür ist SQR, die Quadratwurzel. Kehren Sie zum Programm Kurvenzeichnen oben zurück und verändern Sie der Reihe nach Zeile 20 wie folgt:

```
a)  20 PLOT j, SQR j
b)  20 PLOT j, 2 * SQR j
c)  20 PLOT j, 4 * SQR j
d)  20 PLOT j, 6 * SQR j
e)  20 PLOT j, 8 * SQR j
f)  20 PLOT j, 10 * SQR j
g)  20 PLOT, j, 12 * SQR j
h)  20 PLOT j, 14 * SQR j
```

Sie werden rasch bemerken, daß in aufeinanderfolgenden Diagrammen auf dem Bildschirm alles immer höher steigt – und bei g) bleibt der Computer sogar stehen, weil Grafikpunkte ganz vom Bildschirm verschwinden. Je größer Wert x bei x * SQR j, desto stärker wird die Kurve vertikal gestreckt. Das x ist ein *Normierungsfaktor*, und wenn Sie ihn anpassen, können Sie erreichen, daß grafische Kurvendarstellungen schön auf den Bildschirm passen.

Wenn der Normierungsfaktor zu klein ist, erhalten Sie derart zusammengequetschte Kurven, daß Sie nichts erkennen können. Probieren Sie

```
20  PLOT j, .1 * SQR j
```

Wenn die Funktion, die Sie darstellen, zu groß wird, können Sie sie durch Anpassen des Normierungsfaktors wieder auf den Schirm zurückholen. Beispielsweise verläßt

```
20 PLOT j, j * j
```

den Bildschirm, weil 14 * 14 = 196, was bereits zu groß ist. Die größte Zahl, die Sie darstellen können, ist übrigens 255 * 255 = 65025. Wenn Sie das durch 400 teilen, erhalten Sie 162.5625, was gut in die 175 zulässigen hineinpaßt. Sie erhalten also eine hübsche Kurve, vorausgesetzt, Sie nehmen 1/400 als Normierungsfaktor:

```
20 PLOT j, j * j / 400
```

Es gibt eine ziemlich einleuchtende Grundregel, die dafür sorgt, daß der Normierungsfaktor angemessen gewählt wird. Nehmen wir an, der größte Wert, den die Funktion annimmt, wenn j von 0 bis 255 reicht, sei m (für Maximum). Mit dem Normierungsfaktor s ist dann alles bestens, vorausgesetzt, s * m ist nicht größer als 175 – am besten nahe daran, damit die Kurve nicht zu sehr zusammengequetscht wird. Konkret: Sie können s * m = 175 bestimmen, wenn Sie s = 175/m setzen. (Bei runden Zahlen ist 160 vielleicht besser; für m brauchen Sie weniger eine genaue Zahl als eine vernünftige Schätzung.)

Sie könnten sogar ein Programm schreiben, um m zu berechnen. Wenn wir bei der Funktion j * j bleiben, läßt sich das bewältigen mit:

```
10 LET m = 0
20 FOR j = 0 TO 255
30 LET q = j * j
40 IF q > m THEN LET m = q
50 NEXT j
```

Das zeichnet aber die Kurve noch nicht. Sie fügen also noch die Darstellungsroutine an:

```
60 FOR j = 0 TO 255
70 PLOT j, (175/m) * j * j
80 NEXT j
```

Ein Nachteil: Sie müssen alle Berechnungen *zweimal* machen. Das läßt sich auf eine brauchbare Weise schwer vermeiden, wenn Sie nicht wissen, *welches* j den größten Wert für j * j liefert. In diesem Fall ist das offenkundig j = 255, aber so leicht kann man das nicht immer erkennen. (Sie *können* einen Vektor v (i) von der Größe 256 dimensionieren, die Werte von j * j speichern als v (j * 1) und sie für das PLOT verwenden, aber Vektoren und Arrays verbrauchen viel Speicherplatz! Weiteres zu Vektoren und Arrays finden Sie im Handbuch.)

Sie können nicht nur die vertikale, sondern natürlich auch die horizontale Achse normieren. Die Funktion SQR j oder j * j zeigt das nicht sehr deutlich, also nehme ich 80 + 80 + SIN j, wo das der Fall ist. Nehmen Sie Folgendes:

```
2Ø  PLOT j, 8Ø + 8Ø * SIN (.Ø5 * j)
2Ø  PLOT j, 8Ø + 8Ø * SIN (.1 * j)
2Ø  PLOT j, 8Ø + 8Ø * SIN (.15 * j)
2Ø  PLOT j, 8Ø + 8Ø * SIN (.2 * j)
2Ø  PLOT j, 8Ø + 8Ø * SIN (.25 * j)
```

Jetzt verändert sich der *horizontale* Maßstab – aber der Normierungsfaktor wirkt ganz anders (ist Ihnen das aufgefallen?). Je größer hier der Normierungsfaktor, desto stärker wird die Kurve in waagrechter Richtung zusammengequetscht. Die Kurve weist mehr Krümmungen auf. Warum?

Wenn j von Ø bis 255 reicht, dann reicht die Zahl Ø.5 * j von .Ø5 * Ø bis Ø.5 * 255, also von Ø bis 12.75.

Im zweiten Fall wird also der *doppelte* Bereich von Werten in denselben horizontalen Raum gequetscht.

Mit einem Normierungsfaktor s – das heißt, bei

```
2Ø  PLOT j, 8Ø + 8Ø * SIN (s * j)
```

stellen Sie also den Bereich von Ø bis s * 255 auf der Breite des Bildschirms dar. Je größer s, desto größer der Bereich, umsomehr wird zusammengequetscht.

Wenn Sie also in einem ausgewählten Bereich darstellen wollen, sagen wir, 1ØØØ, brauchen Sie s * 255 = 1ØØØ, das heißt s = 1ØØØ/255. Allgemein gesprochen: Wenn Sie den Bereich Ø bis n wählen wollen, brauchen Sie einen Normierungsfaktor n/255.

Zusammengefaßt:

$$\text{bester Normierungsfaktor vertikal} = \frac{175}{\text{größter darzustellender Wert}}$$

$$\text{bester Normierungsfaktor horizontal} = \frac{\text{oberster Bereichswert der Variablen}}{255}$$

Aufgabe

Wenn Sie nicht wissen, wie etwas am besten aussehen wird, können Sie ein "Dialog"-programm schreiben. Sie wählen damit die beiden Normierungsfaktoren (über INPUT) und können dann die Kurve darstellen; falls Ihnen das Ergebnis nicht gefällt, fahren Sie das Programm noch einmal und ändern den Maßstab.

Schreiben Sie ein solches Programm für die Funktion 8Ø + 8Ø * SIN j.

Hinweis: Wenn h der horizontale und v der vertikale Normierungsfaktor ist, lautet die entscheidende operative Programmzeile

```
2Ø  PLOT j, v * (8Ø + 8Ø * SIN (h * j))
```

Achsen verschieben

Sie fragen sich vielleicht: "Warum die vielen 80 + 80 * SIN j?" Oder auch: "Alles schön und gut, aber wenn die Zahlen nun negativ sind?" Die Antwort auf beide Fragen ist die gleiche. Bei negativen Zahlen muß man die Achsen verschieben, wie Fritz der Fuhrmann einmal sagte.

Probieren Sie dieses Programm aus:

```
10  INPUT s
20  FOR j = 0 TO 255
30  PLOT j, s + SQR j
40  NEXT j
```

Geben Sie s ein mit 0, 10, 20 etc.

Sie erhalten immer dieselbe Kurve; entsprechend dem Wert von s klettert sie aber auf dem Bildschirm immer höher. Das kann man sich in zweifacher Hinsicht vorstellen.

Zum einen: Sie stellen verschiedene Funktionen dar, etwa 5 + SQR j oder 10 + SQR j.

Zum anderen: Sie stellen stets SQR j dar, aber die Position der x-Achse auf dem Bildschirm verändert sich. Siehe Abbildung 20.2, die eigentlich für sich selbst spricht.

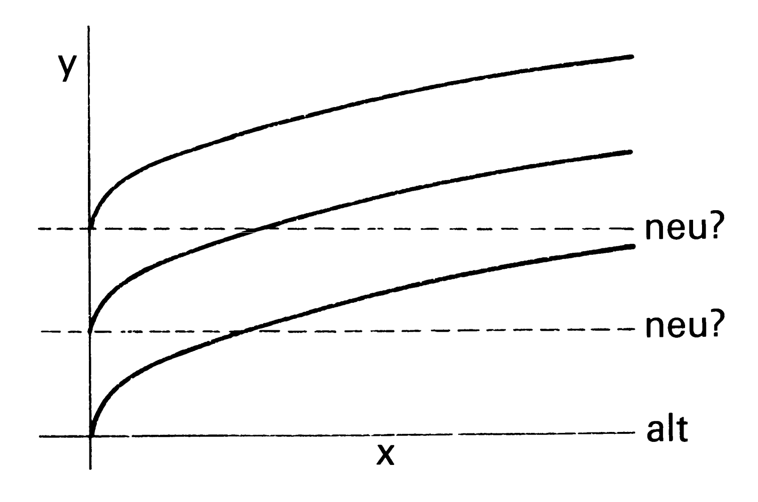


Abb. 20.2 Eine Konstante zu addieren bedeutet dasselbe, wie die x-Achse zu verschieben

Wollen wir etwa eine eindeutige Sinuskurve erhalten, müssen wir das SQR j ersetzen durch 80 * SIN (.1 * j) und s so bestimmen, daß alles auf die Mitte des Bildschirm kommt. s = 80 ist ideal; deshalb die vielen 80 + 80 * SIN.

Damit bewegt sich die x-Achse. Um die y-Achse zu verschieben, können Sie den Bereich von j verändern, von 120 bis 135 statt von 0 bis 255.

Sie können diese Verschiebungen mit Maßstabsverkleinerungen in beiden Richtungen kombinieren und brauchen die Achsen nicht in der Mitte zu halten – das bleibt Ihnen überlassen. Ich habe sogar eine genaue Spezifikation dazu geschrieben, wie man es anstellt, für jede beliebige Kurve in einem bestimmten Bereich das bestmögliche Display auf dem Bildschirm zu erreichen; dann schaute ich mir die Unmenge Algebra an und warf das Ganze wieder weg. Manchmal verstellt Mathematisches den Blick auf das Wesentliche. Das ist einer der Fälle.

Stattdessen schlage ich vor, daß Sie experimentieren, und zwar mit Hilfe des nachstehenden Programms:

```
10  INPUT a, b, c, d
20  PLOT 0, d
30  DRAW 255, 0
40  LET u = – b / a
50  PLOT u, 0
60  DRAW 0, 175
100 FOR j = 0 TO 255
110 PLOT j, c * SIN (a * j + b) + d
120 NEXT j
```

Hier verschiebt a, b, c, d Achsen und Maßstab, während x- und y-Achse eingezeichnet werden. Ich habe nicht gegen Zusammenbrüche geschützt, wenn die Funktion den Bereich überschreitet. Von der Faulheit einmal abgesehen: Damit wird deutlich gemacht, wie notwendig es ist, bei der Wahl von Normierung und Achsenposition vorsichtig zu sein.

(Probieren Sie a = .1, b = – 10, c = 80, d = 80. Im allgemeinen ist ein negativer Wert von b erforderlich.)

Andere Methoden

Es gibt noch andere Methoden, mit PLOT und DRAW Kurven zu zeichnen. Statt dieses Kapitel mit technischen Details zu überfrachten, habe ich Beispiele dafür in den Fertigprogrammen untergebracht. Siehe LISSAJOUS-FIGUREN, SPIRALEN und ROSETTEN und GRAFIKDEMONSTRATION 1 und 2.

21 DEBUGGING V

Manchmal sehen Zahlen gleich aus und sind es gar nicht!

Die Fehler, nach denen wir bisher gesucht haben, stammten aus eigener Produktion und waren, einmal entdeckt, verhältnismäßig leicht zu beheben. Es gibt noch eine andere Art, die durch den Aufbau der Maschine selbst bedingt ist. Dabei handelt es sich nicht um einen Konstruktionsfehler, sondern um eine Folge des Aufbaus aller Computer. Das hängt mit der Genauigkeit zusammen, mit der Computer Zahlen speichern. Wenn wir an alltägliche Verfahren zur Aufnahme von Zahlen denken, liegt nahe, daß die Zahl der Ziffern, die aufgenommen werden kann, begrenzt ist. Beispiel: Der Kilometerzähler eines Autos kann nur 6 Ziffern anzeigen, weil er nur 6 "Fenster" hat. Bei einem Computer ist das nicht anders. Jede Zahl kann nicht mehr besetzen als eine feste Zahl von "Fenstern". Dabei steht aber nicht jedes Fenster für eine Dezimalziffer. Der interne Maschinencode für Zahlen unterscheidet sich ganz grundlegend von der Art, wie wir sie uns vorstellen; mit den grauslichen Einzelheiten will ich Sie nicht behelligen. Die Tatsache, daß eingebaute Ungenauigkeit besteht *und* eine Codeumwandlung stattfindet, bedeutet, daß die äußere Darstellung einer Zahl (wie sie auf dem Bildschirm angezeigt wird) nicht dasselbe sein muß wie die innere. Ich will Ihnen anhand der Logarithmen, wie wir sie von der Schule kennen, ein Beispiel geben. Wenn Sie logarithmisch 2 mit 2 multiplizieren, erhalten Sie:

Zahl	log
2	Ø.3Ø1Ø
2	Ø.3Ø1Ø
3.9999	Ø.6Ø2Ø+

also 2 × 2 = 3.999!

Die Tatsache, daß die Logarithmen nur bis zu 4 Stellen genau sind (also nur 4 Fenster besetzen dürfen), mit der Codeumwandlung *zusammen* (Zahl zu Logarithmus, Logarithmus wieder zu Zahl) ruft die Ungenauigkeit hervor.

Hier ein Programm, das dieselbe Art von Problem hervorruft:

```
1Ø  FOR p = Ø TO .3 STEP .Ø1
2Ø  LET q = ATN (TAN (p))
3Ø  IF p < >q THEN PRINT p, q
4Ø  NEXT p
```

In Zeile 1Ø berechnen wir den Tangens von p und kehren den Prozeß sofort wieder um, indem wir den Arkustangens davon berechnen. Mit anderen Wor-

ten: q müßte denselben Wert enthalten wie p. Zeile 3Ø dürfte demnach nie dazu führen, daß p und q angezeigt werden, weil sie stets gleich sind. Wenn das Programm läuft, erhalten wir folgende Ausgabe:

```
Ø.Ø2    Ø.Ø2
Ø.Ø3    Ø.Ø3
Ø.Ø4    Ø.Ø4
Ø.Ø5    Ø.Ø5
Ø.Ø7    Ø.Ø7
Ø.Ø9    Ø.Ø9
Ø.11    Ø.11
Ø.12    Ø.12
Ø.13    Ø.13
Ø.14    Ø.14
Ø.16    Ø.16
Ø.18    Ø.18
Ø.2     Ø.2
Ø.21    Ø.21
Ø.22    Ø.22
Ø.26    Ø.26
Ø.28    Ø.28
```

Das ist in der Tat ein sehr merkwürdiges Ergebnis, weil der Computer nicht nur Werte anzeigt und damit behauptet, sie wären verschieden, sondern sie auch noch so darstellt, als wären sie gleich! Was ist passiert? Die komplexen mathematischen Prozesse, die hier vorgegangen sind, haben kleine Ungenauigkeiten in der internen Darstellung der Zahlen hervorgerufen; sie führten zu den Unterschieden zwischen p und q. Ungenauigkeiten entstehen aber auch durch die Decodierung des internen Formats zu den auf dem Bildschirm angezeigten Dezimalzahlen. Diese scheinen identisch zu sein, obwohl die Maschine steif und fest behauptet, das seien sie nicht. Beachten Sie, daß bei manchen Werten die internen Codes *in der Tat* identisch sind, etwa für Ø.Ø6 und Ø.Ø8.

Diese Art von Fehler kann außerordentlich verwirrend sein. Manchmal bleibt als einziger Ausweg, in der IF-Anweisung einen kleinen Fehler zuzulassen. Dann hätten wir:

```
IF ABS (p − q) < Ø.ØØØØ1 THEN . . .
```

Die ABS-Funktion ist erforderlich, weil q größer sein könnte als p. Beispiel: Wenn p = 3 und q = 3.1, dann p − q = − Ø.1, was kleiner ist als Ø.ØØØØ1, so daß die Bedingung erfüllt wäre, wenn die ABS-Funktion (die das Minuszeichen kappt) nicht vorhanden wäre. ABS (− .Ø1) = Ø.1, was größer ist als Ø.ØØØØ1, so daß die Bedingung nicht erfüllt ist – und das wollten wir.

22 PROGRAMMIERSTIL

Wenn Sie dann längere Programme schreiben, die Komplizierteres leisten, ist es wichtig, kühlen Kopf zu bewahren. Das Programm "ANTI-RAKETEN-ABSCHIRMUNG" in diesem Kapitel führt zu einem spielbaren Videospiel und *zeigt, was zu verstehen ist unter Programmierstil.*

Gleich vorweg: Das Programm. Wenn Sie wollen, können Sie es als Fertigprogramm behandeln, also einfach abschreiben und mit RUN fahren.

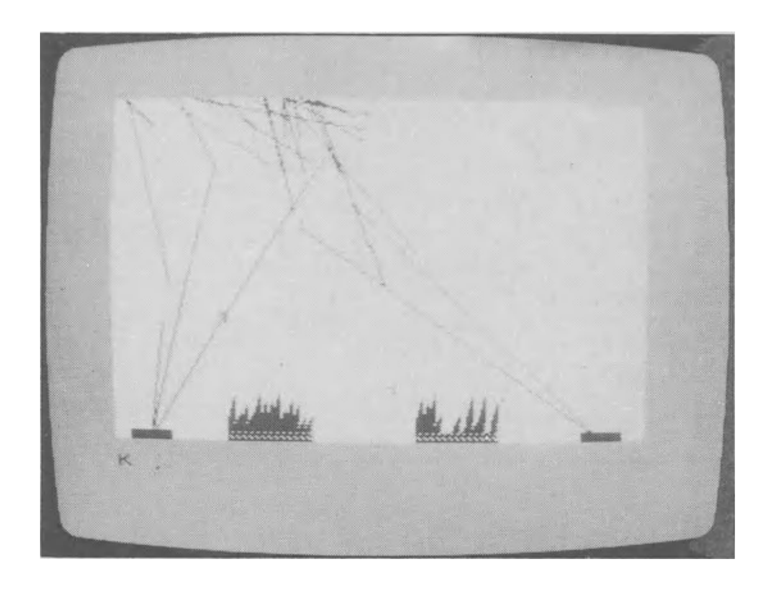
Es fängt damit an, daß es zwei Städte und zwei Raketenabschußbasen zeigt. Die Silos werden von links nach rechts "1" und "2" genannt.

Nun regnen in ziemlich willkürlicher Weise Raketen auf sie herab. Der Weg jeder Rakete besitzt jedoch einen festen Winkel.

Drücken Sie auf der Tastatur "1" oder "2", um eine Ihrer Abschußbasen in Betrieb zu nehmen. (Es kann ein bißchen dauern, bis das Programm darauf reagiert; halten Sie die Taste deshalb so lange gedrückt, bis es das tut.) Sie werden dann aufgefordert, Reichweite und Schußwinkel für die Abfangraketen einzugeben. Wenn Sie eine anfliegende Rakete treffen, wird das dadurch angezeigt, daß der Sprengkopf nicht mehr blinkt.

Früher oder später gehen Ihre Raketen zur Neige (in jeder Basis befinden sich am Anfang 2Ø Stück) oder Ihre Silos oder Städte werden zerstört. Bei dem Spiel kommt es darauf an, möglichst viele Raketen abzuschießen, bevor das eintritt.

Die Werte für "Reichweite" und "Schußwinkel", die Sie eingeben müssen, bedürfen der Erklärung.



Erstens: Der Reichweitenwert benutzt dasselbe Koordinatensystem wie der Spectrum im hochauflösenden Grafikmodus (PLOT). Wenn Sie also etwas treffen wollen, das sich unmittelbar über einem Silo ganz oben am Bildschirm befindet, beträgt die Reichweite 175. Die Gesamtentfernung von der linken unteren bis zur rechten oberen Ecke beträgt etwa 3ØØ.

Zweitens: Der Schußwinkel (in Grad gemessen) wird von der Horizontalen zur rechten Seite der jeweiligen Abschußbasis gelegt. Das heißt: Für Silo 2 liegen die sinnvollsten Schußwinkel zwischen 9Ø° und 18Ø° – also Vorsicht! (Bei Silo 1 liegt allerdings der sinnvolle Bereich zwischen Ø° und 9Ø°.) Eine der Eigenschaften des Programms ist die, daß es Ihnen nicht sagt, ob Sie einen Fehler machen, etwa den, eine Rakete aus dem Bildschirm hinauszuschießen. Es zieht einfach eine Rakete von Ihrem Vorrat ab und zeigt keine Spur der Rakete an.

Hier das vollständige Listing.

ANTI-RAKETEN-ABSCHIRMUNG

```
   1 RANDOMIZE
  1Ø LET display = 5ØØ: LET genmis = 1ØØØ
  2Ø LET movmis = 15ØØ
  3Ø LET fire = 25ØØ: LET prnscr = 3ØØØ: LET hitest = 35ØØ
  4Ø DIM m (3, 2Ø): DIM s (2)
  5Ø LET city = 2: LET s (1) = 2Ø: LET s (2) = 2Ø
  6Ø LET kilsilo = 4ØØØ: LET kilcity = 45ØØ
  7Ø LET score = Ø
  8Ø LET kbhit = 5ØØØ
 1ØØ GO SUB display
 11Ø IF city < 1 OR s (1) + s (2) = Ø THEN GO SUB prnscr
 12Ø GO SUB genmis
 13Ø GO SUB movmis
 16Ø GO TO 11Ø
 5ØØ LET xs = 8
 51Ø FOR I = 1 TO 2
 52Ø FOR x = xs TO xs + 2Ø
 53Ø PLOT x, Ø
 54Ø DRAW Ø, 4
 55Ø NEXT x
 56Ø LET xs = 224
```

```
570  NEXT l
580  LET xs = 56
590  FOR l = 1 TO 2
600  FOR x = xs TO xs + 36 STEP 3
610  LET h = INT (RND*10) + 3
620  FOR y = 0 TO h
630  PLOT x, y
640  DRAW 3, y
650  NEXT y
660  NEXT x
670  LET xs = 144
680  NEXT l
690  RETURN
1000 FOR l = 1 TO 5
1010 LET x = INT (RND*100)
1020 LET a = (RND*PI/2) + .01
1030 FOR p = 1 TO 20
1040 IF m (3, p) = 0 THEN LET m (1, p) = x:
     LET m (2, p) = 175: LET m (3, p) = a: GO TO 1060
1050 NEXT p
1055 GO SUB kbhit
1060 NEXT l
1070 GO SUB kbhit
1080 RETURN
1500 FOR p = 1 TO 20
1505 IF m (3, p) = 0 THEN GO TO 1580
1510 PLOT FLASH 0; m (1, p), m (2, p)
1520 LET xo = 20*COS m (3, p)
1530 LET yo = -20*SIN m (3, p)
1540 IF m (1, p) + xo > 255 OR m (2, p) + yo < 0
     THEN LET m (3, p) = 0: GO SUB hitest: GO TO 1580
```

```
155Ø  DRAW xo, yo
156Ø  LET m (1, p) = m (1, p) + xo
157Ø  LET m (2, p) = m (2, p) + yo
1573  PLOT FLASH 1; m (1, p), m (2, p)
1575  GO SUB kbhit
158Ø  NEXT p
1585  GO SUB kbhit
159Ø  RETURN
25ØØ  INPUT "Entfernung, Winkel:"; rg, ag
25Ø5  IF s (VAL d$) = Ø THEN RETURN
251Ø  IF d$ = "1" THEN LET xb = 18
252Ø  IF d$ = "2" THEN LET xb = 234
2523  PLOT xb, Ø
2525  LET s (VAL d$) = s (VAL d$) - 1
253Ø  LET xf = rg*COS (ag*PI/18Ø)
254Ø  LET yf = rg*SIN (ag*PI/18Ø)
2545  IF fx + xb < Ø OR xf + xb > 255 OR yf < Ø OR yf > 175
      THEN RETURN
255Ø  DRAW xf, yf
256Ø  FOR p = 1 TO 2Ø
257Ø  IF ABS (xf + xb - m (1, p)) > 15 OR ABS (yf - m (2, p)) < 15
      THEN GO TO 27ØØ
261Ø  LET score = score + 1
2625  PLOT FLASH Ø; m (1, p), m (2, p)
263Ø  LET m (3, p) = Ø
27ØØ  NEXT p
271Ø  RETURN
3ØØØ  CLS
3Ø1Ø  IF city < 1 THEN PRINT AT 1Ø, 2; "Staedte zerstoert"
3Ø2Ø  IF s (1) + s (2) = Ø THEN PRINT AT 1Ø, 2;
      "Keine Abwehrraketen mehr"
```

```
3Ø3Ø  PRINT AT 12, 5; score; "Raketen abgeschossen"
3Ø4Ø  GO TO 9999
35ØØ  LET xt = m (1, p) + xo
351Ø  IF xt > 7 AND xt < 29 THEN LET xs = 8: GO SUB kilsilo
352Ø  IF xt > 223 AND xt < 245 THEN LET xs = 224:
      GO SUB kilsilo
353Ø  IF xt > 55 AND xt < 93 THEN LET xs = 56: GO SUB kilcity
354Ø  IF xt > 143 AND xt < 181 THEN LET xs = 144:
      GO SUB kilcity
355Ø  RETURN
4ØØØ  IF xs = 8 THEN LET s (1) = Ø
4Ø1Ø  IF xs = 224 THEN LET s (2) = Ø
4Ø2Ø  PRINT AT 21, xs/8; "□ □ □"
4Ø3Ø  RETURN
45ØØ  LET city = city − 1
451Ø  FOR r = 19 TO 21
452Ø  PRINT AT r, xs/8; "□ □ □ □ □"
453Ø  NEXT r
454Ø  RETURN
5ØØØ  IF INKEY$ = "" THEN RETURN
5Ø1Ø  LET d$ = INKEY$
5Ø15  IF CODE d$ < 49 OR CODE d$ > 51 THEN RETURN
5Ø2Ø  GO SUB fire
5Ø3Ø  RETURN
```

IM EINZELNEN

Die meisten Programme in diesem Buch sind ziemlich kurz und bei ein bißchen Übung recht leicht zu bewältigen. Das obenstehende ist etwas ernsthafter (weniger in der Absicht als im Programminhalt) und lohnt, daß man sich ein bißchen näher damit befaßt, weil die Methode, nach der es geschrieben wurde, eine gebräuchliche und sehr wirksame ist. Man nennt sie *top-down-Prinzip* oder *schrittweise Verfeinerung*.

Der Grundgedanke: Das Programm wird in eine Sequenz von Hauptschritten aufgeteilt. Wir untersuchen dann jeden Schritt und entscheiden, ob er

mühelos direkt codiert werden kann oder aber, ob es lohnt, ihn in kleinere Schritte aufzuspalten. Das setzen wir fort, bis alle Schritte so klein sind, daß sie ohne Schwierigkeiten codiert werden können. Auf jeder Ebene wird ein solcher Schritt zu einer Subroutine.

Bei unserem Programm sieht die erste Stufe der Strukturierung etwa so aus:

Variable indizieren

Städte und Abschußbasen anzeigen

Solange noch Abwehrraketen und Städte vorhanden sind — neue Raketen generieren / vorhandene Raketen bewegen

Stand anzeigen

Wenn Sie sich das Listing ansehen, können Sie erkennen, daß die Initialisierungen zwischen den Zeilen 1 und 8Ø geschehen. Sie rechnen also vielleicht damit, etwas vorzufinden wie

```
1ØØ   GO SUB 5ØØ
```

als Einstieg in die Display-Routine. Wie Sie aber sehen, steht da

```
1ØØ   GO SUB display
```

und dieses "display" ist eine Variable, die im Initialisierungsbereich auf 5ØØ festgesetzt wird. Dadurch wird das Programm leichter lesbar, was besonders bei der Fehlersuche nützlich ist. Beispiel: Sobald das Programm einmal gelaufen ist, können Sie

```
LIST display
```

eintippen, statt sich

```
LIST 5ØØ
```

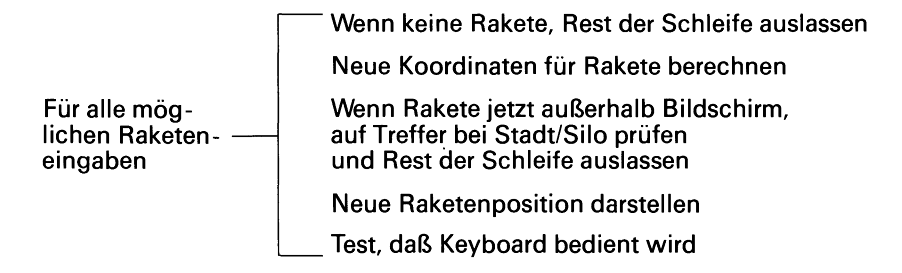
merken zu müssen, wenn Sie sich diesen Programmbaustein ansehen wollen.

Die Zeilen zwischen 11Ø und 16Ø stellen einfach die "während"-Schleife im Gerüst der Programmbeschreibung auf. Sie sehen, daß in der Schleife zwei Unterprogramme aufgerufen werden, eine zur Generierung neuer Raketen (genmis), und eine zur Bewegung von Raketen (movmis), wie Sie erwartet haben werden.

Sehen wir uns zum Exempel "movmis" an. Um eine Rakete bewegen zu können, müssen wir wissen, wo sie ist. Es kann zu jedem Zeitpunkt höchstens 2Ø Raketen geben; in einem Array m sind 3 Informationswerte über jede davon enthalten (x-Koordinate, y-Koordinate und Angriffswinkel). Es wird davon ausgegangen, daß keine Rakete vorhanden ist, wenn ein Angriffswinkel-Fach Null enthält. So brauchen wir von drei Parametern nur einen zu prüfen, um zu erfahren, ob eine Rakete vorhanden ist. Später, wenn wir eine Rakete zerstören

wollen, brauchen wir den Winkel nur auf Null zurückzusetzen, um ihr Hinscheiden anzuzeigen.

Die Aufgliederung innerhalb von "movmis" sieht demnach so aus:



Wenn Sie sich das Listing ansehen, erkennen Sie, daß zwei von den Blöcken als Subroutinen geschrieben sind: der "Test auf Treffer bei Stadt/Silo" (genannt "hitest") und der "Test, ob Keyboard bedient" (genannt "kbhit"). Diese letzte Routine ist da, weil wir wissen müssen, ob der Benutzer eben das Keyboard bedient hat, um eine Abschußbasis in Betrieb zu nehmen.

Sehen wir uns nun kbhit an:

Wenn keine Taste gedrückt, dann RETURN

Wenn gedrückte Taste nicht 1 oder 2, dann RETURN.

Abwehrrakete abfeuern

Sie sehen, daß "fire" (das Abfeuern der Rakete) eine weitere Subroutine ist.

Ist deutlich geworden, daß man mit dieser Methode sich jeweils nur mit einem kleinen Abschnitt befassen muß? Eben das kommt meiner natürlichen Trägheit entgegen.

Ich überlasse es Ihnen, die Gerüste von genmis, fire und prnscr nachzubauen.

REM: ERZÄHL IHNEN, WAS REM IST

Bei der Entwicklung eines Programms ist die REM-Anweisung eine wichtige Gedächtnisstütze. Der Computer ignoriert alle REM-Anweisungen außer bei einem LIST-Befehl. Sie können sich also kleine Hinweise geben, um daran erinnert zu werden, warum ein bestimmter Befehl da ist, wo er steht. Ich habe in diesem Buch allerdings nicht viele REM verwendet, und zwar deshalb nicht, weil alle Programme ohnehin im einzelnen erläutert werden. Wenn Sie aber eigene Programme schreiben, sind REM-Anweisungen ein Himmelsgeschenk. Sie können durch kleine Kniffe REM-Anweisungen in einem Listing hervorheben: Fügen Sie eine Reihe von Sternchen an, also REM******, oder verwenden Sie Steuerzeichen, um sie in einer anderen Farbe anzeigen zu lassen (siehe Handbuch S. 114).

ABWANDLUNGEN UND VERBESSERUNGEN

1 In dem Augenblick, wo Sie eine Abschußbasis in Betrieb nehmen, bestätigt das Programm nicht, welches Silo es ist. Sorgen Sie dafür, daß die gewählte Abschußbasis blinkt (und einige Sekunden nach dem Abfeuern der Rakete abschaltet).

2 Raketen- und Abwehrraketenspuren sehen gleich aus. Geben Sie ihnen verschiedene Farben. (NB: Farbeigenheiten können dazu führen, daß das Ziehen neuer Spuren die Farben der alten verändert. Das läßt sich nicht verhindern, fällt aber nicht ins Gewicht.)

3. Bei verfeinerten Ausgaben dieses Spiels sind die anfliegenden Raketen MIRV-Waffen und zerfallen beim Absturz in Mehrfachsprengköpfe. Das ist nicht schwer zu erreichen, bedarf aber gründlicher Überlegung!

4 Die Zerstörung der Städte geht recht dürr vor sich (sie werden buchstäblich ausgelöscht!). Wie wäre es mit einer Pilzwolke und ein paar Ruinen?

5 Überlegen Sie sich Geräuscheffekte. (Sie verlangsamen das Spiel damit aber stark.)

6 Da kbhit recht selten aufgerufen wird, können Sie "1" oder "2" ziemlich lange drücken, bevor eine Reaktion eintritt. Experimentieren Sie mit anderen Stellen, wo Sie "GO SUB kbhit" im Programm unterbringen können, um die Reaktionszeit zu verbessern, ohne den Rest des Displays allzu auffällig zu verlangsamen.

23 PEEK UND POKE

Mit dem inneren Aufbau des Spectrum brauchen Sie sich in der Regel nicht zu befassen. Wenn Sie wollen, können Sie den Computer aber veranlassen, Ihnen mitzuteilen, was er treibt – und können das für Ihre Zwecke verändern.

Ich könnte eigentlich einen ganzen Band von diesem Umfang über PEEK und POKE allein schreiben. Das Meiste davon wäre viel zu technisch, um Interesse zu erwecken. Diese wichtigen Dinge aber ganz beiseitezulassen, wäre schade. Mit PEEK und POKE können Sie wirklich viel darüber erfahren, *wie* Computer arbeiten. Sie müssen also zuerst einmal eine Vorstellung davon haben, wo Sie anfangen sollen: Wie man dem Spectrum beibringt, von seinen innersten Geheimnissen etwas preiszugeben.

Auf Seite 8 habe ich erwähnt, daß Computer Information als Folgen von Ø und 1 speichern. Jede solche Ø oder 1 wird ein *Bit* genannt (abgekürzt für *b*inary dig*it* = Binärziffer). Sie können sich eine Kette von Ø und 1 als eine Zahl im *Binärsystem* vorstellen – es ist ganz wie das Dezimalsystem, nur verwenden wir anstelle von Einern, Zehnern, Hunderten, Tausendern usw. Einer, Zweier, Vierer, Achter und so fort. Unter den Fertigprogrammen finden Sie eines über Umwandlung Binär/Dezimal, wo das ein bißchen ausführlicher erklärt ist.

Über das Binärsystem brauchen wir eigentlich nichts zu wissen, aber wir *müssen* begreifen, daß Computer mit *Bits* arbeiten. Ihre Bits jonglieren sie in Paketen, die *Bytes* genannt werden. Eine Byte ist eine Folge von acht Bits. So sind 1Ø11ØØØ1 und ØØ111Ø11 typische Bits.

Es gibt 256 mögliche verschiedene Bytes; wenn Sie in Dezimal verwandeln, erhalten Sie die Zahlen von Ø bis 255. Diese Zahlen erkennen Sie vielleicht: die CODE-Werte der Spectrum-Zeichen sind Zahlen zwischen Ø und 255. In der Tat wird jedes Zeichen durch genau ein Byte dargestellt.

Ein Programm (und einige der Schritte, die bei seiner Ausführung geleistet werden) ist lediglich eine Folge von Zeichen, also kann der Computer sie als eine Folge von Bytes speichern. Damit alles in der richtigen Reihenfolge bleibt, gibt er jedem Byte eine Bezugsnummer, genannt seine *Adresse*. Sie sollten sich das Programm also so vorstellen, daß es außerhalb des Computers in einer Form wie dieser existent ist:

Adresse	Byte
1	11ØØ11ØØ
2	Ø111ØØØØ
3	ØØØØØØØØ
4	111Ø1111
.....	

Leider läßt die "Architektur" des Computers ein solches Schema ein bißchen allzu einfach erscheinen.

Der Spectrum besteht aus Mikrochip-Anordnungen mit den Namen

ROM (Read Only Memory = Festspeicher oder Nur-Lese-Speicher)
RAM (Random Acess Memory = Speicher mit wahlfreiem Zugriff oder Direktzugriffsspeicher)
CPU (Central Processing Unit = Zentraleinheit)
SCL (Sinclair Logic Chip = Sinclair Logikschaltung)

ROM speichert den BASIC-Interpreter oder -übersetzer, RAM Ihr Programm und alles, was geleistet werden muß, während es gefahren wird, CPU übernimmt Arithmetik und Logik, und SCL organisiert die Zusammenarbeit der anderen.

CPU oder SCL interessieren uns nicht, um so mehr aber ROM und RAM. Mit PEEK können wir genau feststellen, welche Bytes in welchen Adressen von ROM und RAM gespeichert sind.

Beispielsweise zeigt eine Anweisung

```
PRINT PEEK 837
```

das in Adresse 837 gespeicherte Byte an (es ist 4Ø oder in Binärschreibweise ØØ1Ø1ØØØ). Angezeigt wird die entsprechende Dezimalzahl, so daß es nicht schadet, sich ein Byte als einfach eine Zahl zwischen Ø und 255 vorzustellen.

Wo sollen wir nachsehen, wenn wir mit PEEK sinnvoll umgehen wollen?

Die ROM-Adressen reichen von Ø bis 16383. Es gibt zweifellos gute Gründe, mit PEEK in ROM nachzusehen – Sie können feststellen, wie der Spectrum den Fernsehapparat anweist, ein bestimmtes Zeichen anzuzeigen und damit seine Spielchen zu treiben (etwa, es in vierfacher Größe zu bringen.)

Interessanter sind aber die RAM-Adressen. Sie beginnen bei 16384; diejenigen bis zu 23754 (oder mehr, wenn das Mikrolaufwerk angeschlossen ist) werden von der Maschine gesetzt und verwendet. Ihr Programm kommt in eine Adresse, die in der Systemvariablen PROG gespeichert ist; sie befindet sich in den Adressen 23635 und 23636. Die Startadresse für das Programm ist übrigens

```
PEEK 23635 + 256*PEEK 23636
```

Ohne das Mikrolaufwerk müßte das 23755 sein. Versuchen Sie es und sehen Sie selbst.

Um herauszufinden, was in RAM gespeichert ist, lassen Sie den Spectrum die Arbeit tun. (Machen Sie das immer, wo es geht!) Hier ein Programm dafür:

```
1ØØØ LET q = PEEK 23635 + 256 * PEEK 23636
1Ø1Ø LET r = PEEK q
1Ø2Ø IF r > 23 THEN PRINT q; "□ □"; TAB 12; CHR$ r
1Ø3Ø IF r < = 23 THEN PRINT q; "□ □"; r; TAB 12; "?"
1Ø4Ø LET q = q + 1
1Ø5Ø GO TO 1Ø1Ø
```

Ich habe große Zeilennummern verwendet, weil ein Testprogramm davorgesetzt und festgestellt werden soll, wo und wie es gespeichert wird, indem wir GO TO 1ØØØ verwenden.

Tippen Sie das ein, dazu ein Testprogramm:

```
1Ø  REM start
2Ø  PRINT AT Ø, 1Ø; "*"
```

Geben Sie jetzt GO TO 1ØØØ und passen Sie auf. Sie erhalten diese Anzeige:

```
23755   Ø     ?
23756   1Ø    ?
23757   7     ?
23758   Ø     ?
23759   234   REM
2376Ø   115   s
23761   116   t
23762   97    a
23763   114   r
23764   116   t
23765   13    ?
23766   Ø     ?
23767   2Ø    ?
23768   23    ?
23769   Ø     ?
2377Ø   245   PRINT
23771   172   AT
23772   48    Ø
23773   14    ?
23774   Ø     ?
23775   Ø     ?
23776   Ø     ?
```

Dann will es wissen, ob abgerollt werden soll, aber wir möchten uns ansehen, was auf dem Bildschirm steht.

Wir können unser Programm – jedenfalls den größten Teil davon – in der dritten Spalte entstehen sehen: REM, s, t, a, r, t, . . ., PRINT, AT, Ø, . . . Aber

anderes Zeug ist auch dabei. Die dritte Spalte scheint zum besseren Begreifen nichts beizutragen, aber die zweite tut es. In Adresse 23756 finden wir 1Ø, in 23767 steht 2Ø: Vermutlich sind das die Zeilennummern. (Bei den niedrigeren Zeilennummern ist das ein bißchen eigenartiger, als es aussieht, und die 7 in 23757 und die 23 in 23768 sind in Wahrheit die Zahlen der Zeichen in den betreffenden Zeilen: siehe Handbuch, Kapitel 24. Aber das meiste ergibt doch sofort Sinn.)

Die Fragezeichen in Spalte 3 stehen aus gutem Grund dort: Wenn Sie in Zeile 1Ø2Ø "IF r > 23" weglassen und 1Ø3Ø streichen, läuft das Programm nicht. Das liegt daran, daß diese Zeichen *Steuer*zeichen sind (die beispielsweise Farben setzen). Der Computer versucht sie als Befehle auszuführen, versteht nichts und hält mit einer Fehlermeldung an. Was sie sind, können Sie aber aus der Zeichentabelle im Handbuch, Anhang A (oder Datenblatt) erfahren. Beispiel: In Adresse 23756 haben wir Zeichen Nummer 13, das ENTER ist – ah ja, wir haben es vorhin ja wirklich gedrückt, nicht wahr?

Ein paar Details machen also Kopfzerbrechen, aber Sie sehen doch, wo das Programm gespeichert ist, und da sitzt es wirklich. Damit Sie mehr davon sehen, drücken Sie "y" für ein Abrollen des Bildschirms (eine Einrichtung, die unser PEEK-Programm besonders nützlich macht). Nun erhalten wir

23777	Ø	?
23778	Ø	?
23779	44	,
2378Ø	49	1
23781	48	Ø
23782	14	?
23783	Ø	?
23784	Ø	?
23785	1Ø	?
23786	Ø	?
23787	Ø	?
23788	59	;
23789	34	"
23790	42	*
23791	34	"
23792	13	?

und Sie können die Auflistung nach gelegentlichem Abrollen noch ziemlich lange fortsetzen. Bald listet sich die Routine ab Zeile 1ØØØ auf . . .

Versuchen Sie ein paar andere Programme in den Zeilen 1Ø, 2Ø, 3Ø etc. Lassen Sie die Zeilen 1ØØØ–1Ø4Ø, wie sie sind, und gehen Sie mit GO TO 1ØØØ daran, Ihre Programme über PEEK zu besichtigen. Sie werden in der Auflistung bald Ansätze eines Schemas erkennen.

Ein offenkundig eigentümlicher Punkt ist der, wie *Zahlen* gespeichert werden. Die 1Ø in PRINT AT Ø, 1Ø, zum Beispiel, scheint die Adressen 2378Ø –23787 zu besetzen. Viel Platz für eine so kleine Zahl, möchte man meinen. Das hängt aber mit der Tatsache zusammen, daß der Spectrum *Fließkomma*-Arithmetik beherrscht (Dezimalzahlen in der Art von 23.567) und einen geeigneten Code verwendet. Sie können sich stundenlang allein damit amüsieren, zu verfolgen, wie eine Zahl konkret gespeichert wird.

Aber das wahrhaft Schöne daran ist: Seit Sie wissen, *wo* ein bestimmtes Programmbyte zu Hause ist, können Sie es *verändern*. Die Anweisung, die Ihnen diese schreckliche Macht über den armen Spectrum verleiht, heißt

POKE

Das wollen wir in Aktion sehen.

Was leistet unser kleines Programm von oben? Es zeigt an der Position Ø, 1Ø * an. Geben Sie RUN und sehen Sie selbst.

Gehen wir mit POKE hinein. Fügen Sie zwei Programmzeilen an:

```
3Ø   POKE 2379Ø, 96

4Ø   GO TO 1Ø
```

Fahren Sie es mit der Eingabe

```
GO TO 3Ø
```

Statt * wird Ihnen £ angezeigt.

Warum? Weil Zeile 3Ø in Adresse 2379Ø (wo das ursprüngliche * gespeichert war) mit POKE das neue Zeichen mit Code 96 hineinsetzt, das £ ist.

Wenn Sie der Maschine LIST befehlen, werden Sie feststellen, daß Zeile 2Ø des ursprünglichen Programms jetzt lautet

```
2Ø   PRINT AT Ø, 1Ø; "£"
```

Experimentieren wir noch ein bißchen mehr. Drücken Sie BREAK und nehmen Sie die Zeilen 3Ø und 4Ø heraus. Ändern Sie Zeile 2Ø mit EDIT zu ihrem früheren Wortlaut ab. Machen Sie aus der Ø gleichzeitig eine 1, so daß es jetzt heißt

```
2Ø   PRINT AT 1, 1Ø; "*"
```

Jetzt GO TO 1ØØØ und alles wieder mit PEEK prüfen. Lassen Sie abrollen, wenn der Bildschirm voll ist.

Sie werden die Veränderung bemerken. Aus den Adressen 23772 und 23776 wurde

```
23772    49    1

23776    1     ?
```

Sonst bleibt alles beim alten.

Um unser neues Wissen zu testen, fügen Sie folgende Programmzeilen an:

```
30  POKE 23772, 49
40  POKE 23776, 1
50  GO TO 20
```

Ändern Sie Zeile 2∅ wieder ab zu PRINT AT ∅, 1∅; "*". Drücken Sie RUN.

Der Bildschirm sollte *zwei* Sternchen zeigen, auf den Bildschirmzeilen ∅ und 1. Der Computer geht das erste Programm durch und zeigt auf Bildschirmzeile ∅ an, dann setzt er mit POKE die Werte, die für Bildschirmzeile 1 gebraucht werden; mit GO TO 2∅ muß er danach auf der neuen Zeile erneut anzeigen.

Versuchen Sie LIST; erneut werden Sie entdecken, daß aus Zeile 2∅ wieder PRINT AT 1, 1∅; "*" geworden ist.

In den ROM können Sie mit POKE nicht hineingehen, versteht sich – man kann ihn nur lesen! Ist auch gut, denn ein versehentliches POKE könnte den BASIS-Interpreter ruinieren. Aus diesem Grund dürfen Sie ohne Bedenken nach Lust und Laune mit POKE durch die Gegend fahren, um zu sehen, was sich abspielt.

Die Möglichkeiten, die POKE eröffnet, sind riesengroß. Um damit aber richtig umgehen zu können, müssen Sie mehr über die genauen internen Codes wissen, die der Spectrum verwendet. Kapitel 24 und 25 des Handbuchs liefern die Grundkenntnisse, die Sie brauchen. Wenn Sie sehr eifrig sind, können Sie den Rest mit der PEEK-Routine oben, Zeilen 1∅∅∅–1∅5∅, ergründen. Mehr will ich nicht sagen, weil es besser ist, selbst dahinterzukommen, als eine langatmige Beschreibung zu lesen, die ein anderer verfaßt hat. Wie bei anderen Abschnitten des Buches wollte ich Sie aber in eine sinnvolle Richtung lenken. Bis Sie hier angekommen sind, werden Sie erkannt haben, daß PEEK und POKE nicht die schrecklichen Rätsel sind, als die sie oft hingestellt werden. Sie dringen nur tiefer ein in das *System*, wie der Spectrum funktioniert. Experimentieren Sie mit den beiden – eine faszinierende Herausforderung, auf die sich einzulassen sehr lohnend sein kann.

24 TIPS

Neben ein, zwei Problemen, die immer wieder auftreten, gibt es den einen oder anderen nützlichen und weniger bekannten Kniff. Hier die unverblümte Wahrheit . . .

HINDERNISSE, DIE EINEM ZU SCHAFFEN MACHEN

1. Es ist ganz natürlich, daß man versucht, eine Zahl x dadurch ins Quadrat zu erheben, daß man x ↑ 2 berechnet. Das geht gut bei positiven Zahlen; obwohl aber das Quadrat einer negativen Zahl durchaus Sinn ergibt, funktioniert der nach oben weisende Pfeil hier nicht. (Der Grund: x ↑ a wird berechnet als EXP (a * LN x), und der Logarithmus einer negativen Zahl ist nicht definiert. Nehmen Sie, wenn x vermutlich negativ wird, lieber x * x.

2. Im gleichen Zusammenhang: Wenn x eine ganze Zahl ist, hat x ↑ 2 die unerfreuliche Angewohnheit, nicht ganz genau zu sein. Das kann zu Schwierigkeiten von der Art führen, wie wir sie in Debugging V besprochen haben. Ebenso bei x ↑ 3: Es ist wirklich oft besser, man schreibt x * x * x.

3. Wenn Sie in einer Zeile mit Mehrfachanweisungen REM verwenden, dürfen Sie nicht vergessen, daß *alles*, was danach kommt (in dieser Zeile) vom Computer unbeachtet gelassen wird. So hat

```
1Ø   REM start: LET x = 99
```

überhaupt keine Wirkung. Dagegen setzt

```
1Ø   LET x = 99; REM start
```

x auf 99.

4. Jeder IF/THEN-Befehl wirkt so: Die *ganze* Zeile nach dem THEN wird ausgeführt, vorausgesetzt, die IF-Bedingung ist erfüllt. Trifft das aber nicht zu, führt die Maschine von den Befehlen nach THEN *überhaupt* nichts aus. So setzt

```
1Ø   IF x = Ø THEN LET y = Ø: IF x <> Ø THEN LET y = 1
```

y dann auf Ø, wenn x gleich Ø, wird aber *völlig übergangen*, wenn x < > Ø. Im Gegensatz dazu wird bei den getrennten Zeilen

```
1Ø   IF x = Ø THEN LET y = Ø
2Ø   IF x <> Ø THEN LET y = 1
```

y bei jedem x, das nicht Null ist, auf 1 gesetzt. Falls Ihre bedingten Befehle also schiefgehen, überprüfen Sie zuerst Ihre Mehrfachbefehle!

5. In einer IF/THEN-Anweisung behandelt der Computer den *gesamten* Ausdruck zwischen IF und THEN, bevor er prüft, ob sie wahr ist. Das kann zu Zusammenbrüchen und unbeabsichtigten Fehlermeldungen führen. Beispielsweise verursacht

```
1Ø  IF x$ < > " " AND VAL x$ = 5 THEN STOP
```

dann einen Zusammenbruch, wenn x$ leer ist – obwohl der Bestandteil x$ < > " " falsch ist und daher die ganze Bedingung ohne Rücksicht auf den Wert von VAL x$ als falsch gilt. Der Computer besteht trotzdem darauf, VAL x$ zu berechnen, wenn x$ = " ", was er nicht mag. Sie mögen ja *meinen*, Sie hätten Ihr Programm gegen Zusammenbrüche geschützt, aber das kann ein Irrtum sein . . .

6. Es gibt zwei Möglichkeiten, Leerstellen zu setzen. Einmal mit der SPACE-Taste, zum anderen mit Grafik-8 bei CAPS SHIFT. Das sind für den Spectrum *nicht dieselben Zeichen*. Eine Suche nach Leerstellen mit dem Befehl

```
1Ø  IF n$ (i) = "□" THEN GO TO 5ØØØ
```

scheitert also, wenn die Leerstelle in n$ (i) nicht dieselbe Leerstelle ist wie die in Zeile 1Ø Ihres Programms. Leider können Sie den Fehler in einer Auflistung nicht erkennen!

7. Sie können ein Bild durch SAVE sichern mit

```
SAVE "Picture" SCREEN$
```

Es ist schrecklich einfach, es mit

```
LOAD "Picture"
```

wieder laden zu wollen – dann erscheint zwar die Meldung "Bytes: Picture", aber es rührt sich nichts. Sie hätten natürlich befehlen müssen

```
LOAD "Picture" SCREEN$
```

8. Die Qualität des Farbdisplays hängt in starkem Maß von der Qualität Ihres Fernsehapparats und der Feineinstellung ab. Bei einem billigen tragbaren Farbfernseher kann es sein, daß das Display überhaupt nicht befriedigend ausfällt: Kaufen Sie also *auf keinen Fall* eigens einen kleinen Farbfernseher als Monitor, ohne sich *vorher* zu vergewissern, daß er in Ordnung ist. Wenn Sie im Laden Ihren Spectrum vor dem Kauf nicht anschließen dürfen, dann gehen Sie in ein anderes Geschäft!

9. Wenn Sie versuchen, ein Programm mit LOAD zu laden, ertönt aus dem Lautsprecher ein sehr lauter, schriller Piepton, sobald ein Programmname auftaucht. Wenn Sie diesen Ton hören und auf dem Bildschirm nichts erscheint, ist etwas ganz entschieden nicht in Ordnung.

1Ø. Wenn Sie ein Programm im Speicher haben und es auf Band sichern wollen, aber nicht mehr wissen, an welcher Stelle auf dem Band Sie sind, und außerdem

befürchten, eine zu große Lücke zu lassen oder etwas Wichtiges zu überschreiben, geben Sie ein

```
VERIFY "Kaese"
```

wobei "Kaese" ein Name ist, der für kein Programm verwendet wird. Lassen Sie das Band laufen und beobachten Sie die Meldungen, um festzustellen, wo Sie ungefähr sind. An der richtigen Stelle halten Sie das Band an, drücken BREAK und laden Ihr Programm wie gewohnt mit LOAD.

KNIFFE MIT SYSTEMVARIABLEN

1. Einen weitaus befriedigenderen Tastatur-Knackton können Sie mit der Direkteingabe

```
POKE 236Ø9, 5Ø
```

erzielen. Andere Zahlen als 5Ø führen zu leicht veränderten Ergebnissen, aber 5Ø ist ungefähr richtig.

2. Sie können die automatische Wiederholung durch direkte Eingabe von (etwa)

```
POKE 23562, 2
```

beschleunigen. Verändern Sie für schnellere oder langsamere Wiederholungsautomatik zu 1 oder 3. Wenn sie ganz wegfallen soll, zu Ø.

3. Sie können die Zeit, in der die Maschine vor einer automatischen Wiederholung wartet, durch Eingabe von (etwa)

```
POKE 23561, 2Ø
```

verkürzen. Weniger als 2Ø verkürzt die Verzögerung, mehr erhöht sie.

4. Manche Programme müssen automatisch abrollen. (Viele ZX81-Spiele nutzen das Abrollen, und Sie möchten sie vielleicht auf den Spectrum übertragen.) Einen SCROLL-Befehl gibt es aber nicht.

Sie können dem Ding aber einreden, daß es abrollt, wenn Sie die Routine

```
1ØØØ  PRINT AT 21, Ø
1Ø1Ø  POKE 23692, 2
1Ø2Ø  PRINT
```

verwenden. Überhaupt: Sobald der Bildschirm voll wird, geben Sie POKE 23692 mit einer Zahl größer als 1 und versuchen mit PRINT eine neue Zeile anzuzeigen: Das führt zu einem Abrollen ohne Eingabe von der Tastatur.

5. Der Befehl DRAW x, y zeichnet, fein, wie er ist, von der derzeitigen laufenden Position xØ, yØ zur neuen Position xØ + x, yØ + y. Das heißt, x und y sind die *Versetzungen*, die gebraucht werden, nicht die neuen Koordinaten. Ein Weg, von der laufenden Position zu einer neuen Position x, y zu zeichnen, besteht darin, den Befehl

```
DRAW x-PEEK 23677, y-PEEK 23678
```

zu verwenden. Der Vorteil: Das funktioniert sogar dann, wenn Sie nicht mehr wissen, wo die letzte PLOT-Position war (was sehr leicht vorkommt), und führt nicht zu kumulativen Rundungsfehlern, wenn Sie das in einer Schleife benützen, etwa, um Kurven zu zeichnen. Die Adressen 23677 und 23678 enthalten natürlich die Koordinaten des zuletzt dargestellten Punktes!

25 WOVON ICH IHNEN NICHTS ERZÄHLT HABE

Und nun: Die grauenhafte Wahrheit . . .

In Ihrem Spectrum steckt viel, viel mehr, als ich in diesem Buch auch nur habe andeuten können. Ich hätte liebend gerne mehr aufgenommen, aber dann wäre ein siebzehnbändiges Werk daraus geworden, von dem jeder Band Sie fünfzig Mark gekostet hätte . . . Bis Sie aber *diesen* Band überstanden haben, wird das Handbuch von Sinclair viel verständlicher sein. (Nichts gegen das Handbuch – es hat eben wie alle Handbücher die Verpflichtung, Sie über alles zu unterrichten, was dann auch zu manchmal knappster Darstellung führen muß.)

Beispielsweise habe ich nicht von den mathematischen Funktionen wie EXP, COS, TAN, LN gesprochen, habe Ihnen nichts erzählt über benutzergewählte Funktionen DEF FN A (x, y, z,. . .), über die Bescheid zu wissen sich lohnt, ich habe nur eine Verwendung von USR genannt (aber die anderen führen zum Maschinencode; siehe Kapitel 26 "Was nun?"), ich habe ein paar mehrdimensionale Arrays verwendet, ohne sie zu erklären, ich habe INVERSE nicht erwähnt und nichts zu LOAD und SAVE von mir gegeben, weil das Handbuch sich sehr deutlich dazu äußert und das ganze System ohnehin fast narrensicher ist.

Aber ich möchte hier vor allem eines betonen: Wenn Sie in einem Programmlisting einen Befehl sehen, den Sie nicht verstehen, *können Sie es trotzdem eingeben und fahren*. Sollten Sie auf Abenteuer aus sein, können Sie den rätselhaften Befehl abändern und sich ansehen, was herauskommt: Auf diese Weise kommen Sie vielleicht sogar dahinter, was er bewirkt. Nur Mut.

26 WIE GEHT ES WEITER?

Das ist nicht das Ende, sondern erst der Anfang. Die Frage: Wie geht es weiter?

1 Lesen Sie erneut das Handbuch.
2 Es gibt in Deutschland eine ZX-Usergruppe: Adresse (in ZX-User Zeitung)
3 Auch in anderen Ländern gibt es Usergruppen, etwa in England, zu erreichen unter der Adresse
   Tim Hartnell
   44–46 Earls Court
   London W86EJ
4 Zeitschriften für Heimcomputer gibt es in Hülle und Fülle, vor allem eine ZX81-Userzeitschrift, gegen eine Schutzgebühr erhältlich bei
   Verlag COOPERATION
   Bonderstr. 2
   8000 München 22
5 Ohne jeden Zweifel wird es bald Unmengen von Software (Bänder) und Hardware (Peripheriegeräte) für den Spectrum geben. Lesen Sie die Zeitschriften.
6 Verfolgen Sie die Kritiken über Hardware und Software in der Fachpresse. Die meisten Händler haben einen guten Ruf, aber es gibt auch Cowboytypen, die man am besten meidet, wenn man sie erkennt.
7 Für fortgeschrittenere BASIC-Programmiermethoden oder Maschinencode beim Z8Ø-Mikroprozessor empfehlen wir sehr das Buch "Maschinencode und besseres Basic", verlegt bei, ähm, Birkhäuser, Basel, verfaßt von zwei Herren namens Stewart und Jones, eine mehr als lohnende Anschaffung. Obschon eigentlich für den ZX81 geschrieben, gilt fast alles dort auch für den Spectrum. Für die, die aber alles ganz genau wissen wollen, schlagen besagte zwei Herren und der oben genannte Verlag gleich doppelt zu: "Weitere Kniffe und Programme für den ZX Spectrum" und "ZX Spectrum Maschinencode" erscheinen in Kürze.
8 Wenn Sie ihn nicht schon haben, wird sich der 32K RAM-Zusatzspeicher bald als unwiderstehlich erweisen. Und achten Sie darauf, wann die Mikro-Diskettenlaufwerke auf den Markt kommen . . .

FERTIGPROGRAMME

Die folgenden Programme sollen dazu dienen, verschiedene Eigenschaften des Spectrums anschaulich darzustellen und Ihnen zu zeigen, was man damit alles machen kann. Jedes Programm ist in sich abgeschlossen und braucht nur eingegeben und mit RUN gefahren zu werden. Die Befehle im Listing muß man nicht unbedingt verstehen.

Bis Sie sich durch dieses Buch hindurchgearbeitet haben, sollten Sie aber in der Lage sein, zu analysieren, wie diese Programme funktionieren. Das wird allerdings nicht immer ganz leicht sein; es fällt oft schwer, sich an den Programmierstil einer anderen Person zu gewöhnen. Guter Stil verlangt Klarheit, einesteils deshalb, weil es eine nervenzerrüttende Aufabe ist, Programme abändern zu wollen, die aussehen wie "Der Arzt von Stalingrad" auf Chinesisch, und anderenteils, weil man dann ohnehin weniger Fehler macht.

Ich habe bewußt viele Programme mit unsauberen Zeilennummern stehen lassen. Es ist verblüffend, wie oft sich Fehler einschleichen, wenn man ein Programm neu numeriert. Und ich wollte hervorheben, daß Sie *nicht* Zeilennummern verwenden müssen, die Vielfache von 1Ø sind. Die einzigen Gründe dafür, das zu tun, sind abgesehen von Mode oder Pedanterie, Platz für Veränderungen im Programm oder für das Einfügen von Ablaufüberwachern zu lassen, wenn man auf Fehlersuche geht.

In diesen Listings habe ich die Grafikzeichen so gezeichnet, wie sie auf der Tastatur erscheinen. Kästchen um einen Buchstaben wie [H] bedeuten Negativschrift, ein leeres Kästchen zeigt eine Leerstelle an. Offenkundige Leerstellen sind nicht eigens angezeigt, wohl aber solche, die wichtig und leicht übersehen werden können.

Noch ein Wort zur *Farbe*. Bevor Sie irgendein Programm fahren, können Sie BORDER, PAPER und INK durch direkte Eingabe per Keyboard bestimmen. Beispiel:

```
BORDER 3: PAPER 5: INK 1
```

Ich habe solche Befehle in die Fertigprogramme *nicht* mit aufgenommen – wenn Sie wollen, können Sie sie selbst einfügen. Das Ziel war, Ihnen bei der Eingabe des Programms Zeit sparen zu helfen und die Sache klarer zu gestalten – auch wenn das Display dann nicht ganz so spektakulär ausfällt. Ebenso sind manche Programme ziemlich einfacher Art, und zwar deshalb, weil ich hoffe, daß Sie zu begreifen versuchen werden, was sie *leisten*, und sie nicht einfach blind abtippen (dafür sollten Sie sich Taschenbücher mit Spielen für Spectrum und ZX81 besorgen).

Die Programme haben keine bestimmte Reihenfolge. Also ran!

DIE FLÄCHE EINES DREIECKS

Als erstes Fertigprogramm habe ich eines ausgesucht, dessen interne Vorgänge recht nah an der Oberfläche liegen. Es berechnet die Fläche eines Dreiecks.

Sind die Seiten eines Dreiecks a, b, c, dann wird seine Fläche bestimmt durch die Formel $\sqrt{s (s - a) (s - b) (s - c)}$, wobei $s = 1/2 (a + b + c)$. Dieses Programm benützt die Formel dazu, die Fläche eines Dreiecks zu berechnen, dessen Seiten der Reihe nach mit INPUT eingegeben werden.

```
 1Ø  INPUT a
 2Ø  PRINT "a = □"; a
 3Ø  INPUT b
 4Ø  PRINT "b = □"; b
 5Ø  INPUT c
 6Ø  PRINT "c = □"; c
 7Ø  LET s = .5 * (a + b + c)
 75  LET x = s * (s - a) * (s - b) * (s - c)
 8Ø  IF x < Ø THEN GO TO 11Ø
 9Ø  PRINT "FLAECHE IST □"; SQR x
1ØØ  STOP
11Ø  PRINT "DREIECK KANN NICHT GEBILDET WERDEN"
```

Programmhinweise

1. Programme, die Formeln auswerten und die Lösungen anzeigen, sind so durchsichtig, daß es Schwindel wäre, sie als echte Programme auszugeben! Sie haben eher Ähnlichkeit mit mechanisch ablaufenden Taschenrechnerprogrammen. Auf dieser Stufe kann aber nach meiner Meinung ein so leichtes Programm nicht schaden.
2. Sie können diese Art von Programm leicht so abwandeln, daß jede vernünftige Formel berechnet wird. Vielleicht ist das ein Weg, mathematische Formeln interessanter zu machen und den Spectrum auch zu Lernzwecken zu benützen. Hier ein paar Vorschläge für denkbare Programme:

   a) Die Fläche einer Kugel vom Radius r ist $4\pi r^2$.
   (π ist auf dem Bildschirm PI: Taste M im erweiterten Modus.)
   b) Das Volumen einer Kugel vom Radius r ist $4/3\pi r^3$.

c) Das Volumen eines Zylinders von Radius r und Höhe h ist $\pi r^2 h$.
d) Das Volumen eines Kegels vom Radius r und Höhe h ist $1/3\pi r^2 h$.
e) Die Lösungen der quadratischen Gleichung $ax^2 + bx + c = \emptyset$ werden gegeben durch

$$x = (- b \pm \sqrt{}(b^2 - 4ac)/2a$$

(Berechnen Sie die + und − Wurzeln getrennt. Wenn $b - 4ac < \emptyset$, sind die Wurzeln imaginär. Darauf müssen Sie eingehen, entweder mit PRINT "IMAGINAERE WURZELN", oder indem Sie einen Programmteil erfinden, der sie berechnet, falls Sie sich mit komplexen Zahlen auskennen. Sie müssen auch die Möglichkeit berücksichtigen, daß $a = \emptyset$.)
f) Die Summe $1 + 4 + 9 + \ldots + n^2$ entspricht $1/6n\ (n + 1)\ (2n + 1)$. Geben Sie n durch INPUT ein und zeigen Sie die Summe mit PRINT an. Berechnen Sie das zum Vergleich auch durch Addieren der Serie mit FOR/NEXT, sobald Sie das Kapitel SCHLEIFEN auf Seite 26 gelesen haben.
g) Die Schwingungszeit eines Pendels von der Länge l ist $T = 2\pi\sqrt{}(l/g)$, wobei g die Beschleunigung infolge der Schwerkraft ist: 981 cm/sec².
h) In Wirklichkeit ist g von Ort zu Ort verschieden. Eine genauere Annäherung ist die, daß der Wert vom Breitengrad B und der Höhe h über dem Meeresspiegel abhängt, und zwar nach der Formel

$$g = 98\emptyset.616 - 2.5928 \cos (2\ L) + \emptyset.\emptyset\emptyset69 \cos (2\ L) - \emptyset.\emptyset\emptyset\emptyset3\ h\ \text{cm/sec}^2$$

Schreiben Sie ein Programm zur Berechnung von g und T, wenn L, h und l bekannt sind.

BIN ICH IM MINUS?

Falls Sie sich für Dreiecke nicht begeistern können: Warum nicht ein Programm, mit dem der Stand des Bankkontos berechnet werden kann?

```
 10  PRINT "BISHERIGER KONTOSTAND □";
 20  INPUT b
 30  PRINT b
 40  PRINT "LISTE SOLL"
 50  INPUT d
 60  IF d < 0 THEN GO TO 90
 70  LET b = b − d
 80  GO TO 50
 90  PRINT "LISTE HABEN"
100  INPUT d
110  IF d < 0 THEN GO TO 140
120  LET b = b + d
130  GO TO 100
140  PRINT "JETZIGER KONTOSTAND □"; b
```

Programmhinweise

1 Zeilen 60 und 110 sind *Begrenzer* (siehe Seite 35). Wenn Sie eine negative Zahl eingeben, weiß der Computer, daß Sie die Liste abgeschlossen haben. (Diese negative Zahl wird in Ihren Kontostand *nicht* aufgenommen!)

2 Vergessen Sie neben den notierten Scheckbeträgen die laufenden Ausgaben nicht. (Eine naheliegende erste Verbesserung: sie ins Programm einbauen. Vielleicht gelingt Ihnen das.) Und Entnahmen aus Bankautomaten.

3 Haben: Gehaltsscheck nicht übersehen! Vielleicht läßt sich auch er einbauen.

4 Ihre kleineren Kinder können damit stundenlang Bank spielen.

5 Wenn Sie noch die beiden Zeilen:

```
 55  PRINT d
105  PRINT d
```

einfügen, zeigt der Computer die Einzelposten an. Sind es mehr als 20, ist der Bildschirm voll, und man muß abrollen, um weitermachen zu können.

TIGERJAGD

Tief im finsteren Sinclair-Wald lauert ein gräßlicher Tiger. Können Sie ihn finden?

```
     PAPER 7: INK 8: BORDER 1
10   INPUT "Waldgroeße waehlen: max 16 □"; w
15   IF w > 16 THEN LET w = 16
20   FOR i = Ø TO 8 * w STEP 8
30   PLOT i + 64, 32
40   DRAW Ø, 8 * w
50   PLOT 64, i + 32
60   DRAW 8 * w, Ø
70   NEXT i
80   LET z$ = "Ø1234567891111111"
90   LET y$ = "□ □ □ □ □ □ □ □ □ □ Ø123456"
100  PRINT AT 19, 8; z$ (TO w) + "□" + "x"
110  PRINT AT 2Ø, 8; y$ (TO w)
120  PRINT AT 16 - w, 6; "y"; AT 17 - w, Ø
130  FOR i = 1 TO w
140  PRINT TAB 6 - (w - i > 9); w - i
150  NEXT i
200  LET mx = INT (w * RND)
210  LET my = INT (w * RND)
220  INPUT "Wo ist der Tiger? □"; x; "□"; y
225  IF x > w OR y > w THEN GO TO 22Ø
230  LET d = ((mx - x) * (mx - x) + (my - y) * (my - y)) / w / w
240  LET d = d * 8: IF d > 4 THEN LET d = 4
250  PRINT PAPER d + 2; AT 17 - y, 8 + x; OVER 1; "□"
255  IF mx = x AND my = y THEN GO TO 3ØØ
260  GO TO 22Ø
```

```
300  PRINT AT 17 - my, 8 + mx; OVER 1; "*"
310  PRINT AT 0, 0; FLASH 1; "Erwischt!"
320  FOR i = 1 TO 30
330  BEEP .1 / i, i
340  NEXT i
```

Programmhinweise

1 Sie können jede Waldgröße von 1 bis 16 wählen. Geben Sie mehr ein, erhalten Sie Größe 16. Der Wald wird mit x- und y-Koordinaten unten und an der Seite gezeichnet (Zeilen 20–70 für den Wald, 80–150 für die Koordinaten). Beachten Sie: Der Bildaufbau ist der Hauptteil des Programms!
2 Auf die Frage "Wo ist der Tiger?" müssen Sie zwei Zahlen zwischen 0 und w − 1 eingeben: das sind die Koordinaten Ihrer Vermutung. Zahlen außerhalb dieses Bereichs, die Sie eingeben, werden nicht beachtet.
3 Der Computer zeigt dann an dieser Stelle ein farbiges Quadrat an. Die Farbe liefert einen Hinweis darauf, wie nah Sie sind; rot bedeutet sehr nah, magenta ziemlich nah, grün ziemlich weit weg und gelb ganz weit weg.
4 Sie machen weiter, bis Sie den Tiger entdecken. Dann erhalten Sie zum feierlichen Anlaß ein blinkendes "*", eine blinkende Glückwunschmeldung und was Musikalisches. (Die Musik steht in den Zeilen 320–340. Sie können Sie in Ihren eigenen Programmen verwenden, um dieselben Töne hervorzubringen.) ELTERN können, um den Verstand nicht zu verlieren, diese Zeilen streichen.
5 Drücken Sie RUN für einen neuen Versuch.

Aufgaben

1 Sorgen Sie dafür, daß das rote Quadrat blinkt, wenn Sie wirklich ganz nah am Tiger sind. Das können Sie in Zeile 250 tun. Fügen Sie nach OVER 1 eine Anweisung von der Art

FLASH (d < dieses oder jenes)

ein und experimentieren Sie, um herauszufinden, welcher Wert dieses oder jenes sein sollte, damit etwas Hübsches herauskommt.
2 Verändern Sie die Meldung zu einem unmanierlichen Ausdruck.
3 Verändern Sie die Musik am Ende: Geben Sie eine Auswahl an Melodien, aus der die Maschine willkürlich eine heraussucht, damit es nicht langweilig wird.

KOMPONIST

Sie können selbst Beethoven sein oder der Spectrum – Johann Strauß . . .

Mit diesem Programm kann der Spectrum Musik machen. Die Noten werden der Reihe nach als Einzelbuchstaben oder Einzelbuchstaben mit dem Zeichen # eingegeben, um Halbtöne anzuzeigen. Geben Sie **, um die Notenfolge zu beenden und den Spectrum zu veranlassen, daß er die Melodie spielt.

```
 1Ø  DIM s (7)
 11  DIM n$ (2)
 12  DIM t (15ØØ)
 2Ø  LET s (1) = – 3: LET s (2) = – 1: LET s (3) = Ø: LET s (4) = 2
 3Ø  LET s (5) = 4: LET s (6) = 5: LET s (7) = 7
 35  FOR q = 1 TO 15ØØ
 4Ø  INPUT "Note eingeben"; n$
 42  IF n$ = "**" THEN GO TO 2ØØ
 45  LET  i = Ø
 5Ø  LET p = CODE n$ – 64
 55  IF p > 1Ø THEN LET p = p – 32
 6Ø  IF n$ (2) = "#" THEN LET i = 1
1ØØ  LET t (q) = s (p) + 1
11Ø  NEXT q
2ØØ  FOR r = 1 TO q
21Ø  BEEP Ø.5, t (r)
22Ø  NEXT r
```

Die Einzelheiten dazu, wie das funktioniert, werden im Abschnitt über Arrays erklärt.

Verbesserungen und Abwandlungen

1 Ersetzen Sie Zeile 4Ø durch:

```
4Ø  LET n$ = CHR$ (INT (RND * 7) + 65)
```

Das Programm komponiert nun sein eigenes Musikstück mit genau 1500 Noten. Dabei gibt es keine Halbtöne, so daß das doch recht eintönig klingt.

2 Die Dauer jedes Tons ist mit 0.5 Sekunden festgelegt. Können Sie sich eine Routine ausdenken, die es dem User erlaubt, die gewünschte Dauer jedes Tons gleichzeitig mit der Note selbst einzugeben? Sie brauchen ein weiteres Array für die Aufnahme der Dauerwerte von derselben Länge wie t. (Bei 16K geht mit zwei Arrays von der Länge 1500 der Speicherplatz zur Neige. Machen Sie jedes Array 750 lang.)

3 Wenden Sie eine ähnliche Methode wie unter 1) an, um Zufallsmusik mit zufälliger Tondauer zu generieren.

4 Die Musik Note für Note einzugeben, kann ein bißchen mühsam werden. Können Sie einen Weg finden, um sie als einzige Kette einzugeben?

5 Das könnte ein nützliches Komponierinstrument sein, wenn Einzelnoten leicht zu verändern wären. Können Sie einen Melodie-Editor (Melodi-tor?) erfinden, bei der die Eingabe von, sagen wir, 38,A# die Bedeutung hat: "verändere die 38. Note zu A#"?

6 Befreien Sie sich aus der Zwangsjacke einer einzigen Oktave. Auch hier brauchen Sie mehr Eingaben pro Note. Beispiel: A, 0 könnte heißen "A in der Oktave um das mittlere C" und C#, 2 könnte bedeuten "C#, zwei Oktaven über dem mittleren C".

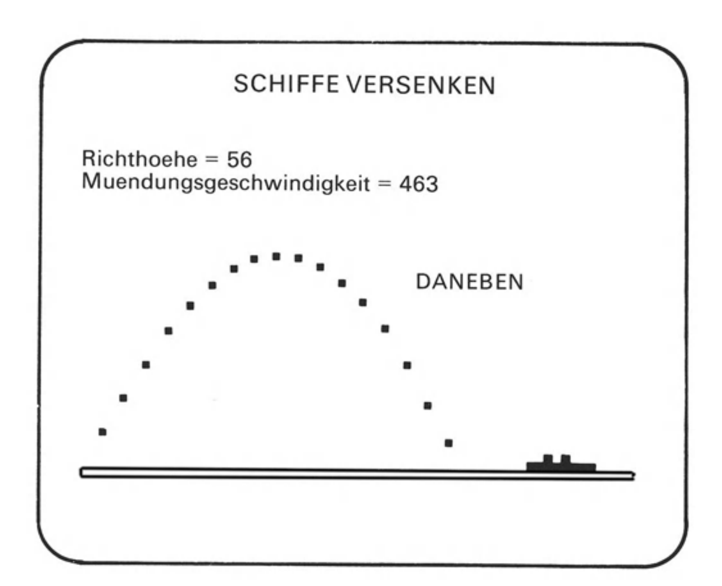
SCHIFFE VERSENKEN

Hier ein Programm, damit die Kinder an einem regnerischen Nachmittag Frieden geben . . .

Eine Patrouillenfahrt in der Nordsee . . . Plötzlich taucht aus dem Nebel das feindliche Schiff auf, rund eine Meile entfernt. Sie befehlen Ihren Kanonieren, Richthöhe und Mündungsgeschwindigkeit an ihrem Geschütz einzustellen. Werden Sie das gegnerische Schiff versenken können?

```
10  PRINT TAB 8; "SCHIFF VERSENKEN"
20  LET t = 15 * (1 + RND)
30  PRINT AT 21, 0; INK 5; INVERSE 1; "□□□□□□□□□
    □□□□□□□□□□□□□□□□□□□□
    □□□" (32 Leerst.)
40  PRINT AT 20, t - 2; INK 4; "▄▙▙▄"
50  INPUT "Richthoehe = □"; e
60  PRINT AT 3, 0; "Richthoehe = □"; e
70  INPUT "Muendungsgeschwindigkeit = □"; v
80  PRINT "Muendungsgeschwindigkeit = □"; v
90  LET a = v * COS (PI * e / 180)
100 LET b = v * SIN (PI * e / 180)
110 FOR j = 0 TO b / 16 STEP .3
115 LET c = .01 * (b * j - 16 * j * j)
120 IF a * j > 6200 THEN GO TO 190
130 IF c > 40 THEN GO TO 170
140 INK 2
150 PLOT .04 * a * j, 4 * c + 8
160 BEEP .005, c + 10
170 NEXT j
180 IF ABS (a * b / 3200 - t) < 3 THEN GO TO 210
190 PRINT AT 10, 20; "DANEBEN!"
200 STOP
```

```
210  FOR j = 0 TO 15
220  PRINT AT 20 - j, t - 2; "Glucker!": BEEP .3, 6 - 3 * j
230  NEXT j
```



Programmhinweise

1 Zuerst wird das Meer gezeichnet, dann eine Schiffssilhouette, die jedesmal zufallsbestimmt anders steht. Zeile 2Ø bestimmt die Zufallsposition, 3Ø zeigt das Meer an, 4Ø das Schiff.
2 5Ø–8Ø geben mit INPUT Richthöhe e und Mündungsgeschwindigkeit v ein und zeigen sie an. Der Spieler muß, sobald ihn der Computer dazu auffordert, diese Werte über die Tastatur eingeben. e ist in Grad eingeteilt und muß zwischen Ø° und 9Ø° liegen, v ist in Fuß pro Sekunde und muß eine positive Zahl sein.
3 9Ø–17Ø berechnen und zeichnen die Flugbahn der Granate in Abständen von Ø.3 Sekunden – siehe die mathematischen Bemerkungen unten.
4 18Ø gibt Ihnen einen TREFFER, wenn Ihre Granate auf dem Wasser innerhalb von 6ØØ Fuß um das Schiffszentrum einschlägt. Wenn das Spiel schwieriger werden soll, verändern Sie die 3 zu 2 oder 1.5, wenn leichter, zu 4 oder 5.
5 19Ø–23Ø sind Ausgaberoutinen für einen Treffer oder einen Fehlschuß.
6 Für Start oder neuerlichen Start RUN gefolgt von ENTER drücken, wie gewohnt.
7 Der Bildschirm ist hier, der Simulation entsprechend, 64ØØ Fuß breit.
8 Das Programm ist halbwegs "benutzerfreundlich" und läuft (dank Zeile 12Ø) sogar dann weiter, wenn die Granate den Bildschirm verläßt.

9 Die Grafikzeichen in Zeile 40 sind Grafik–3 und 1 mit CAPS SHIFT.
10 "PI" in den Zeilen 90 und 100 ist Taste M im erweiterten Modus, *nicht* die Tasten P und I!

Mathematische Bemerkungen

Die Bahn der Granate wird berechnet nach einer mathematischen Formel, die (bei Vernachlässigung des Luftwiderstands) für einen bei Schwerkraft bewegten Körper gilt – unterstellt mit 32 Fuß/sec² (deutscher Wert = 9.75 m). In Zeile 110 steht t für die Zeit; a ist das horizontale Element der Geschwindigkeit, b das vertikale; PI * e / 180 verwandelt Bogengrade in Radianten. Zeile 150 berechnet die Höhe zur Zeit j und verwandelt in Bildschirmkoordinaten (jedes PLOT-Pixel hat 25 Fuß im Quadrat); in Zeile 110 ist b / 16 die Zeit bis zum Aufschlag im Wasser; in Zeile 180 ist a * b / 180 die bis dahin zurückgelegte Entfernung; 3200 steht da, weil wir in Zeile 40 eine PRINT-Anweisung verwenden, so daß die Entfernung bis zum Schiff 200 + t beträgt.

Wenn Ihnen irgendeine andere Eigenschaft des Programms Kopfzerbrechen macht, ändern Sie ab und sehen Sie sich an, was dabei herauskommt.

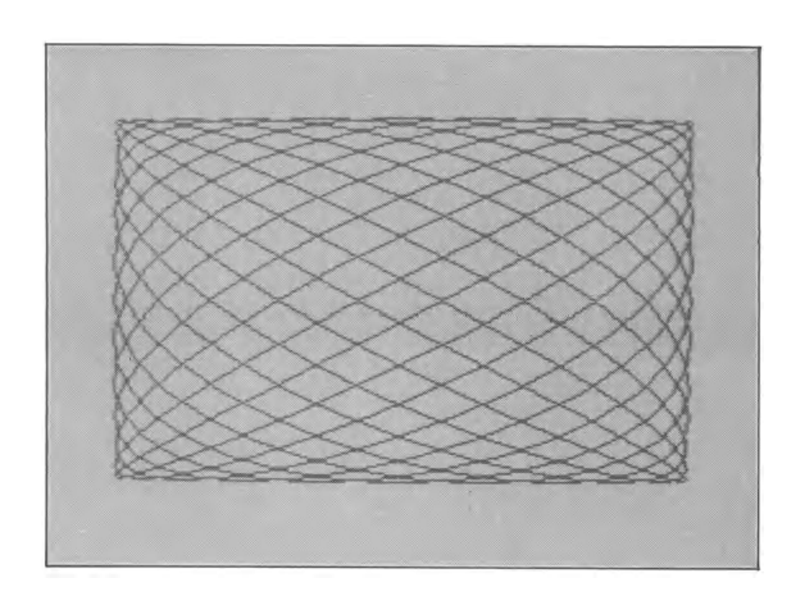
Bedenken Sie Folgendes: Ein Computerspiel wie dieses zu *spielen*, hat zwar nur geringen Lernwert (die richtige Taste drücken, Entfernungen und Winkel schätzen), aber es *schreiben* oder *verstehen*, verlangt bestimmte Kenntnisse in Programmieren und Mathematik und kann als Motivation in der Schule wie zu Hause dienen.

LISSAJOUS-FIGUREN

Hier eine Abwandlung der PLOT-Routine, die wunderschöne Kurven hervorbringt: Lissajous-Figuren.

Sie wurden 1815 von Nathaniel Bowditch erfunden, sind aber nach Jules-Antoine Lissajous benannt, der sie 1857 zum zweitenmal fand. Sie veranschaulichen einen anderen Weg, mit PLOT interessante Kurven zu gestalten. Genutzt wird dabei, was die Mathematiker *Parameterdarstellung* nennen. Das heißt nichts anderes, als daß Sie x und y von einer Variablen t abhängig machen und PLOT x, y für Folgen von t befehlen.

```
 1Ø  INPUT "Erste Zahl"; p
 2Ø  INPUT "Zweite Zahl"; q
 3Ø  INPUT "Phasenverschiebung"; r
 4Ø  LET t = Ø
 5Ø  LET m = p
 6Ø  IF q < p THEN LET m = q
 7Ø  LET x = 127 + 12Ø * COS ( t * PI / 18Ø * p + r)
 8Ø  LET y = 87 + 8Ø * SIN (t * PI / 18Ø * q)
 9Ø  IF t = Ø THEN PLOT x, y
1ØØ  IF t > Ø THEN DRAW x–PEEK 23677, y–PEEK 23678
11Ø  LET t = t + 1Ø / m
12Ø  GO TO 7Ø
```



Programmhinweise

1 Damit es schöne Ergebnisse gibt, sollten p und q ganze Zahlen sein.
2 r kann beliebig sein, das Beste ist aber, wenn es zwischen Ø und 3 liegt.
3 Ein guter Anfang ist p = 5, q = 7, r = 1.
4 PI ist Taste "M" im erweiterten Modus.
5 Wenn Sie haltmachen wollen, geben Sie BREAK + CAPS SHIFT.
6 Wenn es spektakulärer werden soll (wobei das Zeichnen aber länger dauert), probieren Sie p = 31, q = 29, r = Ø.
7 Zeile 1ØØ berechnet die Versetzung vom alten Plotpunkt zum neuen, so daß DRAW zwischen beiden eine Linie ziehen kann. Die Systemvariablen in den Adressen 23677 und 23678, in die mit PEEK hineingegangen wird, enthalten die letzten PLOT-Koordinaten. Dieser Kniff ist überaus nützlich.

MONOPOLY-WÜRFELN

Der Sepctrum kann dank seines Zufallszahlengenerators statistisch arbeiten. Dieses Programm simuliert Monopoly-Würfeln.

Wenn Sie schon *Monopoly* gespielt haben, werden Sie sich erinnern, daß Sie zwei Würfel rollen und das Ergebnis addieren müssen, um zu erfahren, wie weit Sie ziehen dürfen. Bei 36 Würfen können Sie erwarten, daß die Gesamtzahl

2 im Mittel 1mal vorkommt
3 im Mittel 2mal vorkommt
4 im Mittel 3mal vorkommt
5 im Mittel 4mal vorkommt
6 im Mittel 5mal vorkommt
7 im Mittel 6mal vorkommt
8 im Mittel 5mal vorkommt
9 im Mittel 4mal vorkommt
1Ø im Mittel 3mal vorkommt
11 im Mittel 2mal vorkommt
12 im Mittel 1mal vorkommt

Selbstverständlich ergibt sich dieses Verhältnis bei Ihnen nicht genau, aber auf lange Sicht sollten Sie – im Mittel! – nah herankommen.

Sie können mit dem Spectrum die Welt der Statistik erforschen, indem Sie solche Dinge mit Hilfe von Zufallszahlen *simulieren*. Dieses Programm "rollt" 2 Würfel 144mal (um das Dasein zu erleichtern: 4 mal 36), zählt, wie oft die Gesamtzahl 2, 3, 4, . . . etc. ist, stellt das Ergebnis in einem "Balkendiagramm" dar und vergleicht außerdem das tatsächliche Ergebnis mit den theoretisch zu erwartenden Zahlen.

```
1Ø   DIM a (11)
2Ø   FOR j = 1 TO 144
3Ø   LET d = 1 + INT (6 * RND) + INT (6 * RND)
4Ø   LET a (d) = a (d) + 1
5Ø   NEXT j
8Ø   FOR j = 2 TO 12
9Ø   LET q = a (j - 1)
1ØØ  LET q Ø = INT (q / 2)
11Ø  LET q 1 = q - 2 * q Ø
12Ø  FOR t = 1 TO q Ø
13Ø  PRINT AT 18 - t, 4 + 2 * j; "◨"
```

```
140 NEXT t
150 IF q 1 = 1 THEN PRINT AT 17 - q 0, 4 + 2 * j; "▖"
160 NEXT j
170 PRINT AT 19, 8;
    "2 □ 3□ 4 □ 5 □ 6 □ 7 □ 8 □ 9 □ 1 □ 1 □ 1"
180 PRINT AT 20, 24; "0 □ 1 □ 2"
```

Programmhinweise

1 Das Balkendiagramm zeigt die relative Zahl des Vorkommens der Gesamtzahlen 2, 3, 4, . . ., 12. Die theoretische Form ist dreieckig mit dem Gipfel bei 7. Wie nah kommt das wirklich heran? Erhalten Sie dieselbe Form, wenn Sie mit RUN noch einmal laufen lassen? Warum nicht?

ARTIHMETIK-PROBE

Hier ein Beispiel für ein "Lern"-Programm, wenn auch kein sehr ehrgeiziges.

Dieses Programm verlangt vom Benutzer (Ihrem siebenjährigen Kind) eine Addition von zwei zufälligen zweistelligen Zahlen, prüft, ob die Antwort falsch oder richtig ist, und erklärt, wenn sie falsch war, wie man es richtig macht.

```
10  LET v = 10
20  LET c = 0
30  PRINT "Hallo! Ich bin Sinclair Spectrum. Wer bist du?"
40  INPUT e$
50  PRINT "FEIN, □"; e$; "□ kannst du das"
60  PRINT "beantworten?"
70  LET x = INT (v * v * RND)
80  LET y = INT (v * v * RND)
90  IF x + y < = 12 THEN GO TO 7 * v
100 LET x 1 = INT (x / v)
110 LET x 2 = x - v * x 1
120 LET y 1 = INT (y / v)
130 LET y 2 = y - v * y 1
140 IF x 2 + y 2 > = v THEN LET c = 1
150 PRINT x; "+"; y; "□ = □ ? □";
160 INPUT a
170 PRINT a; "□?"
180 LET b = x + y - a
190 PRINT ("richtig" AND b = 0) + ("schade, falsch"
    AND b < > 0)
200 IF b = 0 THEN GO TO 190
210 PRINT x 2; "+"; y 2; "□ ist □"; x 2 + y 2 - c * v;
220 IF c = 1 THEN PRINT "□ uebertrage 1"
```

```
230 PRINT "□ und dann □", x 1; "+"; y 1; "+ den Uebertrag"
    AND c = 1; "□ ist □"; x 1 + y 1 + c
240 PRINT "ergibt □", x + y
```

Programmhinweise

1 Zeile 30 stärkt zuerst einmal das Vertrauen. Sie will vom Benutzer den Namen wissen. Zeile 50 verwendet den Namen dazu, eine Frage zu stellen.

2 Mit Zeile 160 teilt der Benutzer dem Computer mit, was er für die richtige Antwort hält.

3 Die Zeilen ab 200 erklären dem Benutzer, wie die Addition hätte erfolgen sollen, wenn er sie falsch gemacht hat.

4 Die Hauptschwierigkeit bei "Lern"-Programmen dieser Art ist die, daß sie meistens lange Textdisplays brauchen. Dieses Programm könnte stark verbessert werden, beispielsweise, um verschiedene Schwierigkeitsgrade einzuführen, mehr Ratschläge zu geben, wenn mit der Lösung des Benutzers etwas nicht stimmt; man könnte das Programm besser dagegen schützen, daß der Benutzer versehentlich falsche Tasten drückt, und so weiter. Und interessantere Rechenaufgaben als schlichte Additionen wären natürlich auch eine Verbesserung. Trotzdem: Die Grundzüge des Ganzen sind aus diesem kleinen Programm schon ersichtlich.

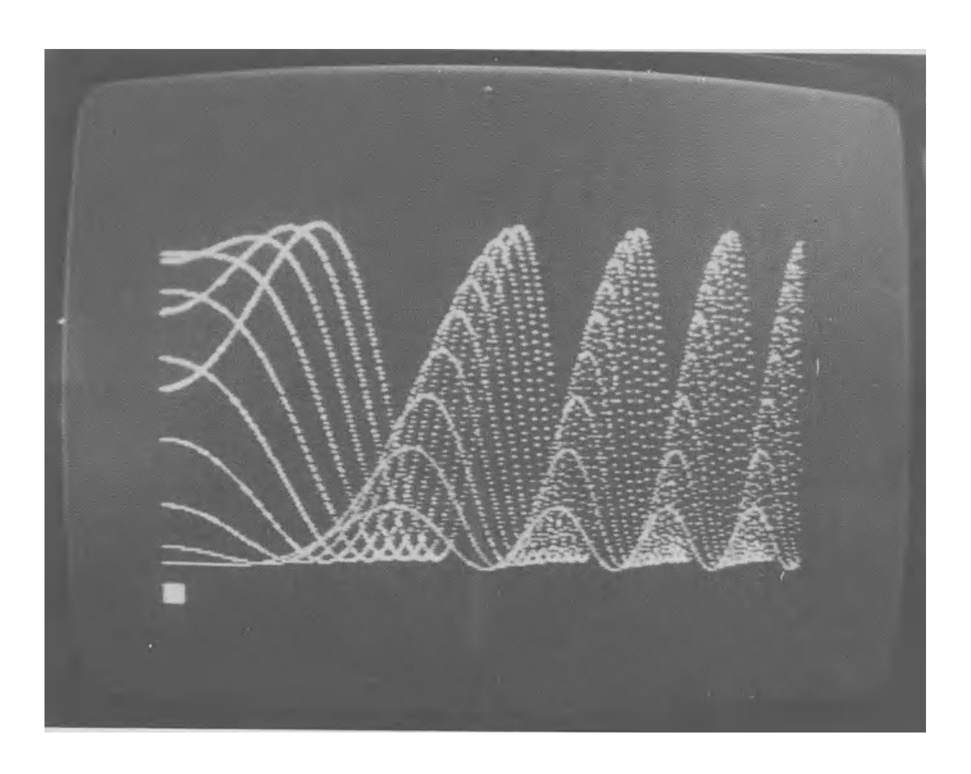
5 Ein gutes Programm müßte nicht jedesmal mit RUN zum Laufen gebracht werden, sondern den Benutzer fragen, ob er es noch einmal versuchen will, und, wenn er ja sagt, automatisch neu laufen. Es ist nicht schwer, ein paar Zeilen anzufügen, um das zu bewerkstelligen.

GRAFIK-VORFÜHRUNG 1

Ein Exempel für die Fähigkeiten des Spectrum bei hochauflösender Grafik

Für ein derart kurzes Programm entsteht hier ein bemerkenswert hübsches Display.

```
    BORDER Ø: PAPER Ø
1Ø  FOR j = Ø TO 9
2Ø  INK 7 − INT (j / 2)
3Ø  FOR i = Ø TO 51Ø
4Ø  PLOT .5 * i, (8Ø + 7Ø * SIN (i * i / 1ØØØØ) + j / 2) * (1 − j * j / 1ØØ)
5Ø  NEXT i
6Ø  NEXT j
```



Programmhinweise

1 Obwohl Veränderungen der INK-Farbe ganze 8x8-Pixelblocks betreffen, ist die Wirkung hier (mit den helleren Farbtönen) recht hübsch, und man bemerkt die Blocks kaum.

2 Das veranschaulicht eine gute Methode, interessante hochauflösende Bilder zu erreichen: Legen Sie mehrere (ähnliche) Kurven übereinander. Mit "ähnlich" meine ich, daß bei jedem Schritt nur ein paar kleine, regelmäßige Veränderungen der Formel stattfinden: Beachten Sie hier, wie die Werte von j die Kurve verändern.

GRAFIK-VORFÜHRUNG 2

Noch ein Angebot von großem Liebreiz

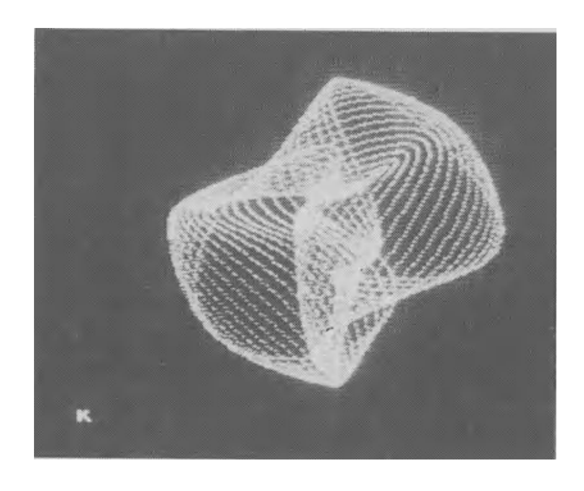
Diesmal zeichnet das dumme Ding ewig weiter, wenn Sie nicht die Geduld verlieren und ihm Einhalt gebieten.

```
    BORDER Ø: PAPER Ø: INK 7
 5  LET i = Ø
1Ø  PLOT 128 + (3Ø + 5Ø * COS (i / 41)) * COS (i / 4Ø),
    88 + (5Ø + 3Ø * SIN (i / 39)) * SIN (i / 4Ø)
15  LET i = i + 1
2Ø  GO TO 1Ø
```

Programmhinweise:

1 Wenn Ihnen das gefällt, versuchen Sie die Zahlen in der Formel zu verändern. Vor allem die 41 und die 39 können ohne Gefahr durch jeden vernünftigen Wert ersetzt werden, ohne daß das Bild vom Fernsehschirm rutscht.

2 Wenn Ihnen Schwarzweiß nicht gefällt, können Sie BREAK geben, dann INK mit diesem oder jenem Wert und schließlich CONTINUE. Die folgenden Pixels werden in der neuen INK-Farbe dargestellt.



PRIMFAKTOREN

Der Spectrum ist ein sehr tüchtiger Zahlenknacker. Nehmen Sie, um sich davon zu überzeugen, eine achtstellige Zahl und verfolgen Sie, wie schnell dieses Programm dafür die Primfaktoren liefert.

```
 1Ø  INPUT k
 3Ø  PRINT k; "□ = □";
 4Ø  LET k Ø = k
 5Ø  IF INT (k / 2) * 2 = k THEN GO TO 14Ø
 6Ø  LET n = 3
 7Ø  IF INT (k / n) * n = k THEN GO TO 11Ø
 8Ø  IF n * n > k THEN GO TO 17Ø
 9Ø  LET n = n + 2
1ØØ  GO TO 7Ø
11Ø  PRINT n; ".";
12Ø  LET k = k / n
13Ø  GO TO 7Ø
14Ø  PRINT "2.";
15Ø  LET k = k / 2
16Ø  GO TO 5Ø
17Ø  IF k = k Ø THEN PRINT FLASH 1; "PRIM"
18Ø  IF k < k Ø AND k > 1 THEN PRINT k
```

Programmhinweise:

1 INT (k / n) * n = k nur dann, wenn k durch n teilbar ist. Das prüft also auf Divisoren.

2 Das Programm versucht k durch 2 und alle ungeraden Zahlen zu teilen, die kleiner sind als die Quadratwurzel von k. Wenn k durch keine davon geteilt wird, ist sie eine Primzahl.

3 Wenn ein Divisor gefunden wird, teilt das Programm k durch ihn und sucht nach einem neuen Divisor der gleichen Größe. Findet er keinen, sucht er nach einem größeren Divisor.

4 Der Computer braucht bei einer achtstelligen Primzahl etwa 35 Sekunden, sonst weniger. (Die angegebene Zeit gilt für 11111117; bei anderen Zahlen ändert sich das naheliegenderweise.)

5 Abwandlungen: Betten Sie das in eine Schleife ein und erzeugen Sie ein Display der Primfaktoren von k, k + 1, k + 2, . . . und so weiter bis unendlich.

6 k kann höchstens 8 Stellen haben, weil der Spectrum größere Zahlen nicht mit ausreichender Genauigkeit bewältigt.

7 Dieses Programm könnte mathematisch ökonomischer gemacht werden, beispielsweise dadurch, daß man Vielfache von 3 oder 5 aus den Prüfdivisoren ausschließt. Können Sie damit ein Programm *schneller* machen? Der Preis für mathematische Straffheit ist erhöhte Programmgröße: gleicht der Gewinn den Verlust aus, wenn Sie vorsichtig sind?

WIE MAN GLEICHUNGEN DRITTEN GRADES LÖST

Programme zur Lösung quadratischer Gleichungen hat jeder. Wie wäre es damit?

Dieses Programm findet alle echten Wurzeln kubischer Gleichungen $ax^3 + bx^2 + cx + d = 0$ mit einiger Genauigkeit.

```
10   INPUT a, b, c, d
50   LET b = b / a
60   LET c = c / a
70   LET d = d / a
80   LET x = 0
90   LET g = 2 * x * x * x + b * x * x - d
100  LET h = 3 * x * x + 2 * b * x + c
110  IF h = 0 THEN GO TO 200
120  IF ABS (x - (g / h)) < 1. E - 8 THEN GO TO 300
130  LET x = g / h
140  GO TO 90
200  LET x = x + 1
210  GO TO 90
300  PRINT "x 1 = □", x
400  LET a = b + x
410  LET b = x * x + b * x + c
420  LET d = a * a - 4 * b
430  IF d < 0 THEN PRINT "Andere imaginaer"
440  IF ABS d < 1. E - 7 THEN PRINT
     "Numerische Instabilitaet moeglich"
445  IF d < 0 THEN STOP
450  PRINT "x 2 = □"; (- a + SQR d) / 2
460  PRINT "x 3 = □"; (a - SQR d) / 2
```

Programmhinweise

1 Das Programm findet eine Wurzel der Gleichung durch ein Iterationsverfahren, das *Newton-Raphson*-Methode heißt. Dabei wird für x ein Versuchswert angenommen und dieser nacheinander verbessert, bis er nah genug an eine Wurzel herankommt. Das bewirken die Zeilen 9Ø–13Ø.

2 Wenn Sie sehen wollen, wie diese Iteration vor sich geht, fügen Sie an

```
131   PRINT x
```

damit Sie sehen, wie die Zahlen sich der Lösung annähern.

3 Sobald das Programm eine Wurzel gefunden hat, teilt es durch sie, um zu einer quadratischen Gleichung zu gelangen, die es durch die Formel in den Zeilen 4ØØ–46Ø löst.

4 Hat diese Gleichung keine echten Wurzeln, gilt Zeile 43Ø; das Programm bricht dann in Zeile 45Ø zusammen, was aber keinen Schaden anrichtet.

5 Eine Feinheit: Für manche kubischen Gleichungen entstehen zu große Ungenauigkeiten in der Arithmetik, und das Programm behauptet vielleicht "Andere imaginär", wenn das gar nicht der Fall ist. Zeile 44Ø weist den Benutzer darauf hin, daß das möglich sein kann. Der Haken dabei: Das Vorzeichen von d ist von entscheidender Wichtigkeit, und wenn d nahe bei Ø liegt, können Fehler eine katastrophale Wirkung haben.

6 Probieren Sie beispielsweise INPUT a = 4, b = − 8, c = 5, d = − 51. Sie müßten erhalten x 1 = 3, andere imaginär. Versuchen Sie jetzt a = 4, b = − 16, c = − 5, d = 51: Diesmal erhalten Sie x 1 = 3, x 2 = 2.62132Ø3, x 3 = 1.62132Ø3.

SPIELAUTOMAT

Auch Sie können sich ein elektronisches Spielgerät gönnen . . .

```
10   LET c = 0
20   DIM a (3)
30   FOR t = 1 TO 3
40   LET r = INT (3 * RND)
50   LET a (t) = r
60   LET i = 5
70   LET j = 10 * t - 5
80   GO SUB 300
90   NEXT t
100  LET c = c - 1
110  IF a (1) = a (2) AND a (2) = a (3) THEN GO TO 700
120  GO SUB 1000
130  INPUT "Stop?"; b$
140  IF b$ = "s" THEN STOP
150  GO TO 30
300  PRINT INK 2 * r; AT i, j; "▚ ▚"; AT i + 1, j; "▚ ▚"
310  BEEP .1, 3 * r
320  RETURN
700  LET c = c + 9
720  GO TO 120
1000 IF c > = 0 THEN PRINT AT 18, 5;
     "Sie gewinnen □"; c; "□ Mark □"
1010 IF c < 0 THEN PRINT AT 18, 5;
     "Sie verlieren □", - c; "□ Mark"
1020 RETURN
```

Programmhinweise

1 Das Display zeigt drei Dekorquadrate. Wenn alle drei gleich sind, gewinnen Sie 9 Mark. Wenn nicht, verlieren Sie 1 Mark. Drücken Sie zum Spielen jede beliebige Taste außer "s" – wir empfehlen ENTER, weil das einen Tastendruck erspart. Wenn Sie aufhören wollen, drücken Sie Taste "s".
2 Im Mittel sollten Sie weder gewinnen noch verlieren. Immer noch besser als am Münzspielgerät in Ihrer Kneipe.
3 Beachten Sie die Verwendung von Subroutinen, um die drei möglichen Quadratarten darzustellen.
4 Beachten Sie den richtigen Gebrauch des Bildschirms: Sie erhalten ein vernünftiges Display in der Mitte, nicht ein winziges, das in einer Ecke zusammengedrängt ist.

OBSKURE FAKULTÄTEN

Der Spectrum erlaubt, daß Sie eine Subroutine aus seinem Inneren herausholen. Man nennt das rekursives Programmieren. Stellen Sie fest, ob Sie dahinterkommen, wie dieses Programm funktioniert.

```
 10  LET f = 1
 20  INPUT n
 30  LET m = n
 40  GO SUB 100
 50  PRINT m; "□ Fakultaet ist □"; f
 70  STOP
100  IF n < = 1 THEN RETURN    ┐
110  LET f = f * n             │
120  LET n = n - 1             ├ Subroutine
130  GO SUB 100                │
140  RETURN                    ┘
```

TAGEFINDER

An welchem Wochentag sind Sie geboren? An welchem Wochentag ist Goethe gestorben? Das können Sie jetzt mit dem Tagefinder feststellen.

Dieses Programm nimmt als Eingabe ein Datum t.m.j an, wo t die Zahl des Tages ist, m der Monat, und j das Jahr (z. B. 23.7.1Ø66) und errechnet wird, welcher Tag das ist (war, sein wird).

```
1Ø   LET a$ = "Ø33614625Ø35"
2Ø   LET b$ = "SONMONDIEMITDONFRESAM"
3Ø   INPUT t
4Ø   PRINT t; ".";
5Ø   INPUT m
6Ø   PRINT m; ".";
7Ø   INPUT j
8Ø   PRINT j; "□ ist □";
9Ø   LET z = j − 1
1ØØ  LET c = INT (z / 4) − INT (z / 1ØØ) + INT (z / 4ØØ)
11Ø  LET x = j + t + c + VAL a$ (m) − 1
12Ø  IF m > 2 AND (j = 4 * INT (j / 4) AND j <> 1ØØ * INT (j / 1ØØ) OR
     y = 4ØØ * INT (j / 4ØØ) THEN LET x = x + 1
13Ø  LET x = x − 7 * INT (x / 7)
14Ø  PRINT b$ (3 * x + 1 TO 3 * x + 3)
```

Programmhinweise

1 Zeile 1Ø speichert eine Liste von zwölf "Monatskorrekturzahlen" in kompakter Form als String. Zeile 1ØØ bestimmt den Monat in der Liste als Teil einer Berechnung, die in Anmerkung 4 erklärt wird.

2 Zeile 2Ø wendet einen ähnlichen Kniff an, um die Anzeigeroutine zu vereinfachen. Beachten Sie, wie Zeile 14Ø die richtige Gruppe von 3 Buchstaben auswählt.

3 Zeile 1ØØ berechnet, wie viele Schaltjahre seit dem Jahre schon vergangen sind. Zur Erinnerung: Vielfache von 4 sind Schaltjahre, nicht aber Vielfache von 1ØØ, wenn sie nicht zugleich Vielfache von 4ØØ sind.

4 Zeile 11∅ ist das Wesentliche bei der Berechnung. Es geht darum, die Zahl der Tage zu berechnen, die seit einem (im Grunde willkürlich gewählten) Datum vergangen sind; um jedoch Platz zu sparen, werden Vielfache von 7 ausgelassen, weil sie auf den Wochentag keinen Einfluß haben. Die Zahl der Jahre j wird also mit 365 multipliziert, aber 365 = 7 * 52 + 1, also verwenden wir j statt 365 * j. Die Eigenheiten der Monatslängen werden berücksichtigt durch a$ (m). Beachten Sie die Verwendung von VAL, um ein einstelliges Zeichen in eine wirkliche *Zahl* zu verwandeln. Um das Programm richtig einzustellen, habe ich ein Datum genommen, bei dem wir den Wochentag kennen – der 3∅.9.81 war ein Mittwoch – und bin dadurch auf die Korrektur von – 1 am Ende gekommen. Zeile 12∅ paßt die Berechnung im Januar oder Februar eines Schaltjahres an, wo sie sonst fehlgehen würde.

5 Eine Kalendereigenheit fehlt. 1585 übernahmen die meisten katholischen Länder den Gregorianischen Kalender, 17∅∅ folgte das evangelische Deutschland. Mein Programm geht vom Kalender "neuen Stils" aus.

Aufgaben

Ändern Sie mit Daten "alten Stils" für die Zeit vor 17∅∅ beziehungsweise 1585 ab (es wurden 12 Tage dazwischengeschaltet).

6 Das Programm nimmt unmögliche Daten an wie –37.12.–992. Ändern Sie es so ab, daß es nur vernünftige Daten akzeptiert.

7 Für Daten v. Chr. funktioniert es nicht. Richten Sie es so ein, daß es das tut. NB: Es hat zwischen v. Chr. und n. Chr. kein Jahr ∅ gegeben, obwohl das von der Logik her erforderlich wäre!

UMWANDLUNG BINÄR/DEZIMAL

Strings und Zahlen sind manchmal enger verwandt, als Sie meinen mögen – so in: Umwandlung Binär/Dezimal.

Binärzahlen verwenden nur Ø und 1; statt Ø, 1, 2, 3, 4, 5, 6, . . . geht das Ø, 1, 1Ø, 11, 1ØØ, 1Ø1, 11Ø, . . . Im allgemeinen wird eine Zahl wie 11Ø1ØØ1 von links nach rechts aufgeschlüsselt als

1 x 1 + Ø x 2 + Ø x 4 + 1 x 8 + Ø x 16 + 1 x 32 + 1 x 64 = 1Ø5

wobei jede Ziffer das Doppelte der vorhergehenden zählt. Computer verwenden für interne Arbeit natürlich Binärzahlen (die modernen Anlagen allerdings in der Praxis verfeinerte Abwandlungen davon).

Die folgenden beiden Programme verwandeln eine mit INPUT eingegebene Zahl von Binär in Dezimal oder umgekehrt.

Binär zu Dezimal

```
1Ø  INPUT a$
2Ø  PRINT a$;
3Ø  LET l = LEN a$
4Ø  LET s = VAL a$ (1)
5Ø  FOR j = 2 TO 1
6Ø  LET s = 2 * s + VAL a$ (j)
7Ø  NEXT j
8Ø  PRINT "□ ist □"; s; "□ in dezimal"
```

Dezimal zu binär

```
1Ø  LET c$ = ""
2Ø  INPUT a
3Ø  PRINT a;
4Ø  LET d = INT (a/2)
5Ø  LET r = a – 2 * d
6Ø  IF r = Ø THEN LET b$ = "Ø"
```

```
 70  IF r = 1 THEN LET b$ = "1"
 80  LET c$ = b$ + c$
 90  IF d = 0 THEN GO TO 120
100  LET a = d
110  GO TO 40
120  PRINT "□ ist □"; c$; "□ in binaer"
```

Programmhinweise:

1 Der INPUT für die Umwandlung binär/dezimal muß eine Folge von 0 und 1 sein, etwa 11000101011. Bei dezimal/binär muß er eine ganze Zahl sein, zum Beispiel 3427006.

2 Die beiden Programme sind so abgefaßt, daß sie verschiedene Wege zur Lösung des Umwandlungsproblems erkennen lassen. Man könnte sie in der Struktur einander sehr annähern.

3 Beachten Sie bei der Umwandlung dezimal/binär, daß die Zeilen 60–70 *nicht* ersetzt werden können durch

```
60  LET b$ = "r"
```

Sie können vermeiden, daß Sie zwei Zeilen schreiben müssen, wenn Sie schlau sind und CHR$ verwenden, oder Sie können schreiben

```
60  LET b$ = ("1" AND r) + ("0" AND 1 - r)
```

Das funktioniert aus Gründen, die damit zusammenhängen, wie der Spectrum Logik bewältigt – vergleiche dazu das Handbuch, Kapitel 13. Oder aber

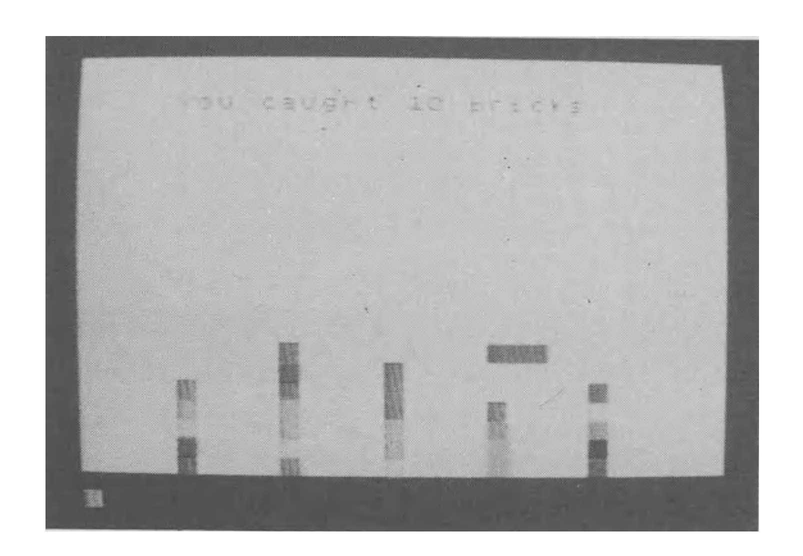
```
60  LET b$ = "0"
70  IF r = 1 THEN LET b$ = "1"
```

bewältigt das auch und spart ein bißchen Platz.

ZIEGELSTEINE

Ein grafisches und bewegendes Erlebnis

Vom Himmel fallen farbige Ziegelsteine. Wenn Sie die mit Ihrer Keule treffen, verschwinden sie, wenn nicht, stapeln sie sich. Wenn die Stapel zu hoch werden, ist das Spiel aus. Wieviele können Sie abfangen?



```
     BORDER 1: PAPER 7
 1Ø  LET h = 11
 2Ø  LET c = Ø
 3Ø  DIM a (5)
 4Ø  LET p = INT (5 * RND + 1)
 5Ø  LET j = 5 * p
 6Ø  FOR i = 1 TO 21 – a (p)
 7Ø  PRINT AT i, j; INK 6 * RND; INVERSE 1: "□"
 8Ø  BEEP .Ø2, 35 – 1.9 * i – j / 2
 9Ø  PRINT AT i – 1, j; "□"
1ØØ  LET m$ = INKEY$
```

```
110  IF m$ = "5" THEN LET h = h - 1
120  IF m$ = "8" THEN LET h = h + 1
130  IF h < 1 THEN LET h = 1
140  IF h > 26 THEN LET h = 26
150  PRINT AT 15, h; "□"; INK 2; INVERSE 1; "□ □ □";
     INVERSE 0; "□":
160  IF i = 15 AND ABS (h + 2 - j) < = 1 THEN LET c = c + 1:
     GO TO 40
170  NEXT i
180  LET a (p) = a (p) + 1
190  IF a (p) = 7 THEN PRINT AT 2, 5; INK 1;
     "Sie haben □"; c; "□ Ziegel erwischt": STOP
200  GO TO 40
```

Programmhinweis

1 Benützen Sie die Pfeiltasten 5 und 8, um die Keule nach links oder rechts zu bewegen.

SPIRALEN UND ROSETTEN

Noch einmal Grafik, und ein nützlicher Kniff für die Verwendung von DRAW, um Kurven darzustellen: Spiralen und Rosetten.

```
    BORDER 1: PAPER 2: INK 7
10  FOR t = 0 TO 4 STEP .5
15  PLOT 128, 88
20  FOR y = 0 TO 720
30  LET x = y * PI / 180
40  LET r = 1.5 * x
50  LET a = 128 + 5 * r * COS (x + t)
60  LET b = 88 + 4 * r * SIN (x + t)
70  DRAW a–PEEK 23677, b–PEEK 23678
80  NEXT y
```

Programmhinweise

1 "PI" in Zeile 30 ist Taste M in erweitertem Modus, nicht die Tasten P und I.

2 Zeile 70 nutzt die Systemvariable COORDS (siehe Handbuch, Kapitel 25), um die richtige Versetzung für das Zeichnen von der alten PLOT-Position zur neuen zu berechnen. Ganz allgemein: Wenn Sie den Befehl PLOT p, q durch DRAW p–PEEK 23677, q–PEEK 23678 ersetzen, zeichnet der Computer eine gerade Linie zur nächsten PLOT-Position. Es gibt noch andere Wege, diese Wirkung zu erzielen, aber die Verwendung von COORDS verhindert, daß Fehler kumulieren, was bei manchen anderen Methoden nicht der Fall ist. Sie können in dieser Hinsicht experimentieren, selbst wenn Sie PEEK nicht verstehen.

3 Wenn Sie statt SPIRALEN lieber ROSETTEN zeichnen wollen, verändern Sie die Zeilen 20 und 40 zu

```
20  FOR y = 0 TO 360
40  LET r = 20 + SIN (7* x)
```

4 Stellen Sie Experimente an und nehmen Sie statt "7" andere Werte wie 3, 4, 5, 6, 8, 9, 10.

PICASSO

Dieses Programm hilft Ihnen vielleicht nicht dabei, so gut zu zeichnen wie Picasso, aber solange Sie bei Farbe 1 bleiben, können auch Sie eine blaue Periode haben!

Das Programm beginnt damit, daß es einen Modus verlangt. Das ist entweder "d" für draw (zeichnen), "2" für erase (löschen) oder "f" für finish (abschließen). Im letzten Fall wird das Programm beendet.

Wenn Sie "d" eingeben, verlangt das Programm eine Farbe. Sie geben sie als die entsprechende Ziffer ein (1 für blau, 2 für rot, etc.).

Bei "d" oder "e" verlangt das Programm nun eine Option, die eine Zahl im Bereich 1 bis 9 sein muß. Die Wirkung jeder Option wird unten erklärt. Beachten Sie, daß das Wort "draw" dann, wenn Modus "e" herrscht, durch "erase" ersetzt werden muß.

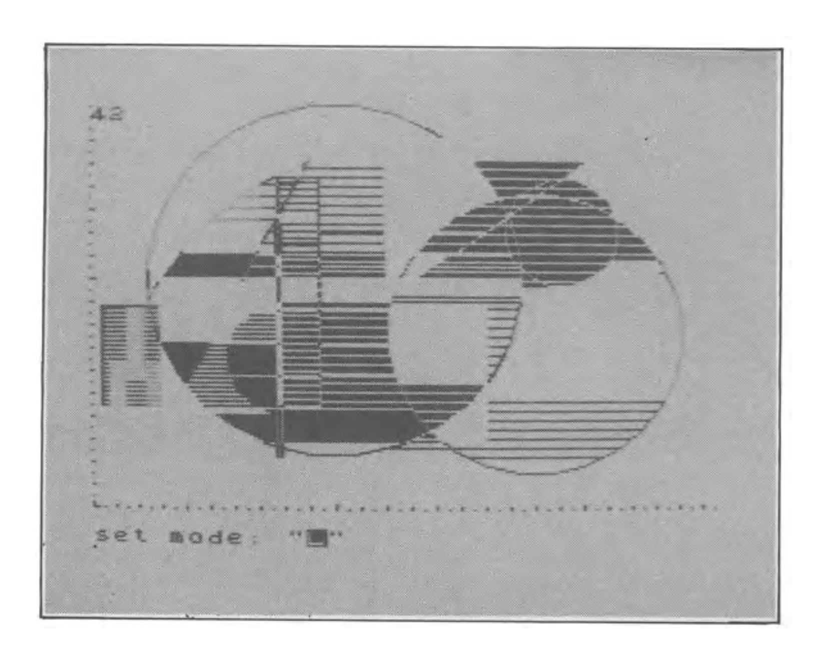
Option	*Wirkung*
1	Zeichnet horizontale und vertikale Skalen als Anhaltspunkt. In jeder fünften Spalte und Reihe steht ein Punkt, in jeder zehnten ein doppelter Punkt (kurzer Strich) in jeder hundertsten ein etwas längerer Strich.
2	Rechteck. Sie sollen die linken und rechten Spalten und untersten und obersten Reihen angeben, wo die Kanten erscheinen sollen. Sie werden dann gefragt, ob das Rechteck gezeichnet oder als Rahmen verwendet werden soll (siehe Option 7).
3	Kreis. Sie sollen Spalte und Reihe angeben, wo der Mittelpunkt erscheinen soll, sowie den Radius.
4	Kreissegment. Sie sollen die Koordinaten der beiden Ecken des Kreissegments und die größte Entfernung von der Kurve zur geraden Linie angeben.
5	Gerade Linie. Sie sollen die Koordinaten der beiden Endpunkte der Linie angeben.
6	Gewölbte Linie. Sie sollen die Koordinaten der beiden Endpunkte der Linie und die größte Entfernung von der Kurve zur imaginären geraden Linie zwischen diesen beiden Punkten angeben.
7	Schattierung. Sie sollen angeben, ob das Gebilde innerhalb des derzeitigen Rahmenrechtecks geschwärzt oder schraffiert werden soll. Beachten Sie, daß das derzeitige Rahmenrechteck das letzte ist, das entweder gezeichnet oder als Rahmen gesetzt wurde. In diesem Zusammenhang wird das Wort "innen" als einschließend verwendet. Mit anderen Worten: Wenn Sie ein Rechteck zeichnen und dann Option 7 aufrufen, wird dieses Rechteck schattiert, es sei denn, darin befindet sich noch ein anderes Gebilde, worauf seltsame Dinge geschehen. Wenn Sie "schraffieren" wählen, werden Sie um Angabe gebeten, wie weit die Linien voneinander entfernt sein und ob die Schraffierungen gerade oder gewölbt sein sollen. Falls gewölbt, sollen Sie die größte Entfernung zwischen der Kurve und der imaginären geraden Linie zwischen den beiden Endpunkten angeben. (Geben Sie zuerst Option 2 ein.)

8 Save. Das sichert ein Bild auf Band. Sie sollen einen Namen angeben.
9 Load. Das lädt ein Bild vom Band zurück. Sie sollen seinen Namen angeben.

Hinweis

Es ist nicht notwendig, das ganze Programm auf einmal zu realisieren. Die Subroutinen, die mit den Zeilen 1000, 2000, 3000 und so weiter beginnen, betreffen die Optionen 1, 2, 3 etc. Wenn Sie also mit etwas Einfachem anfangen und nur Rechtecke und Kreise zeichnen möchten, brauchen Sie die Zeilen ab 4000 nicht einzugeben. Falls Sie es so machen, dürfen Sie natürlich nicht die Optionen 4–9 eingeben, weil Sie sonst eine Fehlermeldung erhalten. Und noch eines muß man beachten: die Routinen, die bei 4000, 6000 und 7000 beginnen, rufen alle die Routine bei 9500.

```
10  INPUT "Modus eingeben:"; m$
20  IF m$ = "f" THEN STOP
30  IF m$ = "e" THEN OVER 1
40  IF m$ = "d" THEN OVER 0: INPUT "Farbe:"; c: INK c
50  INPUT "Option:"; op
60  GO SUB 1000 * op
70  GO TO 10
```



```
1000  FOR x = 0 TO 255 STEP 5
1010  PLOT x, 0
1020  IF x / 10 = INT (x / 10) THEN PLOT x, 1
1030  IF x / 100 = INT (x / 100) THEN PLOT x, 2
1040  NEXT x
1050  FOR y = 0 TO 175 STEP 5
1060  PLOT 0, y
1070  IF y / 10 = INT (y / 10) THEN PLOT 1, y
1080  IF y / 100 = INT (y / 100) THEN PLOT 2, y
1090  NEXT y
1100  RETURN

2000  INPUT "linke Spalte Rechteck:"; lc
2010  INPUT "rechte Spalte:"; rc
2020  INPUT "unterste Reihe:"; br
2030  INPUT "oberste Reihe:"; tr
2040  INPUT "zeichne (d) oder Rahmen (f):"; m$
2050  IF m$ = "f" THEN RETURN
2060  PLOT lc, br
2070  DRAW 0, tr - br
2080  DRAW rc - lc, 0
2090  DRAW 0, br - tr
2100  DRAW lc - rc, 0
2110  RETURN

3000  INPUT "Mittelpunkt Kreis; (Spalte dann Reihe):"; cc, cr
3010  INPUT "Radius:"; r
3020  CIRCLE cc, cr, r
3030  RETURN

4000  INPUT "eine Ecke Segment; (Spalte dann Reihe):"; c1, r1
4010  INPUT "andere Ecke; (Spalte dann Reihe):" c 2, r 2
```

```
4Ø2Ø  INPUT "max. Entfernung Kurve zu Linie:"; d
4Ø3Ø  GO SUB 95ØØ
4Ø4Ø  PLOT c1, r1
4Ø5Ø  DRAW c 2 - c 1, r 2 - r 1
4Ø6Ø  DRAW c 1 - c 2, r 1 - r 2, a
4Ø7Ø  RETURN

5ØØØ  INPUT "ein Endpunkt; (Spalte dann Reihe):"; c 1, r 1
5Ø1Ø  INPUT "anderes Ende; (Spalte dann Reihe):"; c 2, r 2
5Ø2Ø  PLOT c 1, r 1
5Ø3Ø  DRAW c 2 - c 1, r 2 - r 1
5Ø4Ø  RETURN

6ØØØ  INPUT "ein Ende Kurve; (Spalte dann Reihe):"; c 1, r 1
6Ø1Ø  INPUT "anderes Ende; (Spalte dann Reihe):"; c 2, r 2
6Ø2Ø  INPUT "max. Entfernung von der Geraden:"; d
6Ø3Ø  GO SUB 95ØØ
6Ø4Ø  PLOT c 1, r 1
6Ø5Ø  DRAW c 2 - c 1, r 2 - r 1, a
6Ø6Ø  RETURN

7ØØØ  INPUT "schwaerzen (b) oder schraffieren (h):"; m$
7Ø1Ø  IF m$ = "b" THEN LET s = 1: LET a = Ø: GO TO 7Ø7Ø
7Ø2Ø  INPUT "Breite Schraffierung:"; s
7Ø3Ø  INPUT "gerade (s) oder gewoelbt (c):"; m$
7Ø4Ø  IF m$ = "s" THEN LET a = Ø: GO TO 7Ø7Ø
7Ø5Ø  INPUT "max. Entfernung von der Geraden:"; d
7Ø7Ø  FOR r = br TO tr STEP s
7Ø8Ø  FOR c = lc TO rc
7Ø9Ø  IF POINT (c, r) = 1 THEN GO TO 712Ø
71ØØ  NEXT c
```

```
7110  GO TO 7190
7120  LET c 1 = c
7130  FOR c = rc TO 1 c STEP - 1
7140  IF POINT (c, r) = 1 THEN GO TO 7160
7150  NEXT x
7160  LET c 2 = c
7170  PLOT c 1, r
7175  IF m$ = "c" THEN LET r 2 = r: LET r 1 = r: GO SUB 9500
7180  DRAW c 2 - c 1, 0, a
7190  NEXT r
7200  RETURN

8000  INPUT "Namen fuer Sicherung Bild eingeben:"; p$
8010  SAVE p$ SCREEN$
8020  RETURN

9000  INPUT "Namen für Laden Bild eingeben:"; p$
9010  LOAD p$ SCREEN$
9020  RETURN

9500  LET 1 = 0.5 * SQR ((c 2 - c 1) * (c 2 - c 1) + ( r 2 - r 1) *
      (r 2 - r 1))
9510  LET a = ASN (2 * l * d / (d ↑ 2 + l ↑ 2)) * 2
9520  RETURN
```

Abwandlungen und Verbesserungen

1. Im Augenblick sind keine Tests dabei, die dafür sorgen, daß der Benutzer nicht versucht, über den Bildschirm hinauszuzeichnen. Fügen Sie solche ein.
2. Manche Befehle haben die Wirkung, die horizontale Skala zu zerstören, so daß Sie sie mit Option 1 wiederherstellen müssen. Wie ließe sich das vermeiden?
3. Wie wäre es mit einer Routine für vertikale (oder sogar diagonale) Schraffierung?

LINETTE

In Karl Kassierers Kasino wird nicht gewöhnliches Roulette gespielt, da spielt man Linette.

Das ist Roulette, in einer geraden Linie gespielt.

```
 1Ø  LET w = 1ØØ
 2Ø  INPUT "Faites vos jeux □"; b$
 3Ø  CLS
 4Ø  INPUT "Hoehe Einsatz □"; a
 5Ø  LET w = w - a
 6Ø  FOR n = Ø TO 9
 7Ø  PRINT AT 1Ø, 2 * n + 6; CHR$ (48 + n)
 8Ø  NEXT n
 9Ø  LET r = INT (5 * RND) + 5
1ØØ  LET d = INT (1Ø * RND)
11Ø  LET r = r * 1Ø + d
12Ø  FOR n = Ø TO r
13Ø  LET x = n - 1Ø * INT (n / 1Ø)
14Ø  PRINT AT 1Ø, 2 * x + 6; CHR$ 143
145  BEEP .Ø5, x
15Ø  PRINT AT 1Ø, 2 * x + 6; CHR$ (48 + x)
16Ø  NEXT n
17Ø  PRINT AT 1Ø, 2 * x + 6; FLASH 1; CHR$ (48 + x)
18Ø  IF b$ (1) = "g" THEN GO TO 21Ø
19Ø  IF b$ (1) = "u" THEN GO TO 22Ø
2ØØ  IF VAL b$ (1) = d THEN LET w = w + 1Ø * a
2Ø5  GO TO 24Ø
21Ø  IF INT (d / 2) = d / 2 THEN LET w = w + 2 * a
215  GO TO 24Ø
22Ø  IF INT (d / 2) <> d / 2 THEN LET w = w + 2 * a
```

```
240  PRINT AT 16, 9; w; "□ Chips □"
250  IF w > = 0 THEN GO TO 20
260  PRINT "Morgen frueh erscheinen Harrys Schlaeger!"
```

Programmhinweise

1. Nach RUN können Sie entweder auf GERADE, UNGERADE oder eine Zahl zwischen 0 und 9 setzen, indem Sie Ihre Wette eingeben. (Jedes Wort, das mit g beginnt, wird als "gerade" gelesen, jedes, das mit u anfängt, als "ungerade".)
2. Dann teilen Sie mit, wieviel Sie setzen wollen, indem Sie eine Zahl eingeben. Sie fangen an mit 100 Mark, die in Zeile 100 gesetzt werden.
3. Wenn Sie gewinnen, erhalten Sie bei g oder u den doppelten Einsatz, das Zehnfache auf eine Zahl. Theoretisch ist das Spiel dadurch "fair".
4. Sie können mehr setzen, als Sie in der Kasse haben, aber sehen Sie sich vor!
5. Wenn Sie bei einer Buchstabeneingabe aufhören wollen, drücken Sie DELETE und dann STOP. Bei numerischer Eingabe genügt STOP.

LINETTE IM KREIS

Inzwischen ist zwei Häuser weiter Karls alter Konkurrent Georg Geldschneider auf eine originelle Idee gekommen . . .

Das ist Linette, im Kreis gespielt. Hmmmm . . .

```
 1Ø  LET w = 1ØØ
 3Ø  CLS
 4Ø  CIRCLE 123, 91, 84
 5Ø  CIRCLE 123, 91, 6Ø
 6Ø  FOR n = Ø TO 9
 7Ø  PRINT AT 1Ø + 9 * COS (n * PI / 5), 15 + 9 * SIN (n * PI / 5);
     CHR$ (48 + n)
 8Ø  NEXT n
 82  INPUT "Faites vos jeux, Kleiner!"; b$
 84  INPUT "Wieviel willste denn setzen?"; a
 86  LET w = w - a
 9Ø  LET r = INT (5 * RND) + 5
1ØØ  LET d = INT (1Ø * RND)
11Ø  LET r = r * 1Ø + d
12Ø  FOR n = Ø TO r
13Ø  LET x = n - 1Ø * INT (n / 1Ø)
14Ø  PRINT AT 1Ø + 9 * COS (x * PI / 5), 15 + 9 * SIN (x * PI / 5);
     CHR$ 143
145  BEEP .Ø5, x
15Ø  PRINT AT 1Ø + 9 * COS (x * PI / 5), 15 + 9 * SIN (x * PI / 5);
     CHR$ (48 + x)
16Ø  NEXT n
17Ø  PRINT AT 1Ø + 9 * COS (x * PI / 5), 15 + 9 * SIN (x * PI / 5);
     FLASH 1; CHR$ (48 + x)
18Ø  IF b$ (1) = "g" THEN GO TO 21Ø
```

```
190  IF b$ (1) = "u" THEN GO TO 220
200  IF VAL b$ (1) = d THEN LET w = w + 10 * a
205  GO TO 240
210  IF INT (d / 2) = d / 2 THEN LET w = w + 2 * a
215  GO TO 240
220  IF INT (d / 2) <> d / 2 THEN LET w = w + 2 * a
240  PRINT AT 10, 10; w; "□ Chips □ □"
250  IF w >= 0 THEN GO TO 82
260  PRINT FLASH 1; "Gehst du zu Schorsch, ist's Geld im . . . Eimer!"
```

Programmhinweise

1 Das geht genauso wie vorher bei LINETTE.
2 PI ist Taste M in erweitertem Modus.
3 Georg Geldschneider strahlt ein bißchen mehr als Karl Kassierer.

MORSEN AUTOMATISCH

Dit-dit-dit; dä-dä-dä; dit-dit-dit . . .

Das ist ein Beispiel dafür, wie man BEEP verwenden kann. Das Programm akzeptiert eine Meldung über die Tastatur und verwandelt sie in Morsecode. Meldung und Code werden auf dem Bildschirm angezeigt; gleichzeitig kommen die kurzen und langen Töne aus dem Spectrum-Lautsprecher.

```
  5  CLS
 1Ø  INPUT "Eingeben Mitteilung □"; m$
 2Ø  FOR i = 1 TO LEN m$
 3Ø  LET c = CODE m$ (i)
 4Ø  IF c < 32 OR c > 32 AND c < 65 OR c > 9Ø AND c < 97 OR
     c > 122 THEN GO TO 12Ø
 5Ø  IF c > 96 THEN LET c = c - 32
 6Ø  LET c = c - 64
1ØØ  IF c = - 32 THEN PAUSE 4Ø: PRINT "□"
11Ø  IF c > Ø THEN PRINT CHR$ (c + 64); "□";: GO SUB 2ØØ
12Ø  NEXT i
13Ø  STOP
2ØØ  LET k$ = a$ (c)
21Ø  LET t = 1
22Ø  IF k$ (t) = "1" THEN PRINT INK 6; ".";: BEEP .1, 1Ø
23Ø  IF k$ (t) = "2" THEN PRINT INK 3; "-";: BEEP .3,1Ø
24Ø  IF k$ (t) = "□" OR t = 4 THEN PRINT "□": RETURN
25Ø  LET t = t + 1: GO TO 22Ø
5ØØ  REM Eingaberoutine
51Ø  DIM a$ (26, 4)
52Ø  FOR r = 1 TO 26
53Ø  INPUT a$ (r)
54Ø  NEXT r
```

Programmhinweise

1 Um das Array zu setzen, das den Morsecode speichert, müssen Sie mit GO TO 5ØØ anfangen. Sie geben dann über INPUT 26 Strings aus 1- und 2-Zeichen ein: Das sind die Morsecodes für die Buchstaben A–Z, wobei 1 für einen Punkt und 2 für einen Strich steht. Hier der Reihe nach die Strings, die eingegeben werden müssen:

Buchstabe	diesen String eingeben	Buchstabe	diesen String eingeben
A	21	N	21
B	2111	O	222
C	2121	P	1221
D	211	Q	2212
E	1	R	121
F	1121	S	111
G	221	T	2
H	1111	U	112
I	11	V	1112
J	1222	W	122
K	212	X	2112
L	1211	Y	2122
M	22	Z	2211

Geben Sie die Buchstaben nicht ein; sie stehen nur da, damit Sie wissen, wie weit Sie gekommen sind.

2 Nach dem Setzen von a$ drücken Sie GO TO 5. Drücken Sie *nicht* RUN – Sie löschen sonst alle Variablen, die Sie eben so mühselig eingegeben haben. Bei diesem Programm dürfen Sie nie RUN verwenden.

3 Wenn eine Nachricht angefordert wird, geben Sie eine solche ein. Falls sie länger ist als 22 Zeichen, müssen Sie abrollen. Das Programm behandelt Groß- und Kleinbuchstaben gleich und beachtet alles andere nicht. Beispielsweise könnten Sie "Hallo" eingeben.

4 Der Computer zeigt nun die Nachricht in Buchstaben und Morsezeichen an und piept zu den Punkten und Strichen.

5 Beachten Sie, wie der Morsecode in ein Stringarray aufgenommen wird. Das ist eine sehr häufige Verwendung von Arrays: als "Nachschlagetafel" für die Umwandlung von einem Codesystem zum anderen.

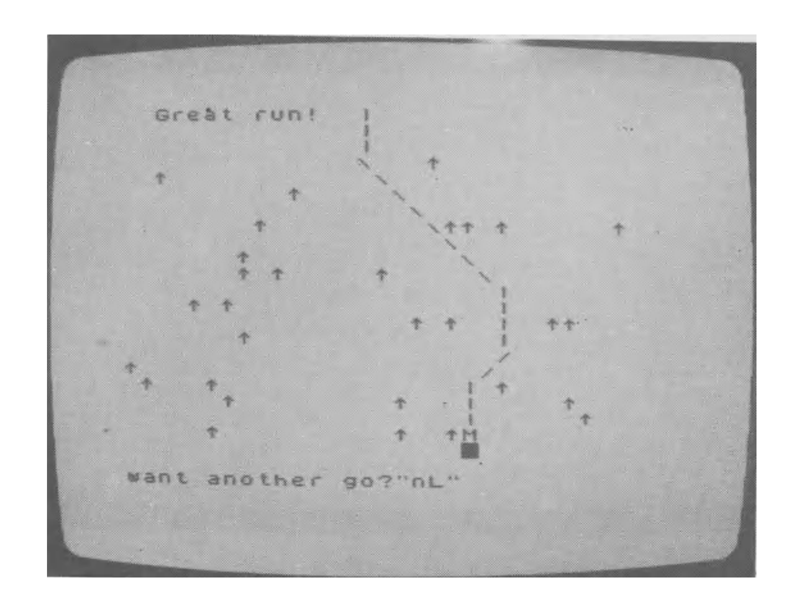
Aufgabe

Ändern Sie das Programm so ab, daß es neben Buchstaben auch Zahlen verwerten kann. Der Morsecode für die Zahlen Ø bis 9 ist der Reihe nach:
–––––, ·–––––, ··–––, ···––, ····–, –····, ––···, –––··, ––––.

ST. MORITZ

Gehen Sie mit dem Spectrum skifahren – zu Hause, wo Ihnen nichts passieren kann.

Sie müssen durch den Wald abfahren und den Bäumen ausweichen, um den Skilift zu erreichen. Sie sind ein "M" (ein Skifahrer in "Ei"-haltung, aus der Luft gesehen – kapiert?) und der Skilift ist ein schwarzer Klecks (warum, weiß ich auch nicht). Es geht nur bergab. Wenn Sie nichts tun, fallen Sie also einfach hinunter, bis Sie ganz unten sind – oder an einen Baum krachen. Sie drücken "z", um nach links, und "m", um nach rechts auszuweichen.




```
1   CLS: RANDOMIZE
2   LET n = 30: LET sc = 14
10  FOR i = 1 TO n
20  LET r = INT (RND * 18) + 3
30  LET c = INT (RND * 32)
40  PRINT AT r, c; "↑"
50  NEXT i
```

```
60   PRINT AT 21, 20; "■"
70   PRINT AT 0, sc; "M"
80   PAUSE 200
90   FOR r = 1 TO 20
100  LET d$ = INKEY$
103  IF d$ = "" THEN PRINT AT r − 1, sc; "|"
104  IF d$ = "m" THEN PRINT AT r − 1, sc; "\": LET sc = sc +1
105  IF d$ = "z" THEN PRINT AT r −1, sc; "/": LET sc = sc − 1
120  IF SCREEN$ (r, sc) = "↑" THEN PRINT AT r, sc; "rrums!":
     GO TO 200
130  PRINT AT r, sc; "M"
135  PAUSE 2
140  NEXT r
150  IF sc < 22 AND sc > 18 THEN PRINT FLASH 1; AT 0,2;
     "Toll gemacht!": GO TO 200
160  PRINT AT 0, 3; "Sie muessen zu Fuss zum Skilift"
200  INPUT "noch einmal?"; q$
210  IF q$ = "ja" THEN GO TO 1
220  CLS: PRINT AT 10, 2; "Et maintenant l'apres-ski . . ."
```

Abwandlungen und Verbesserungen

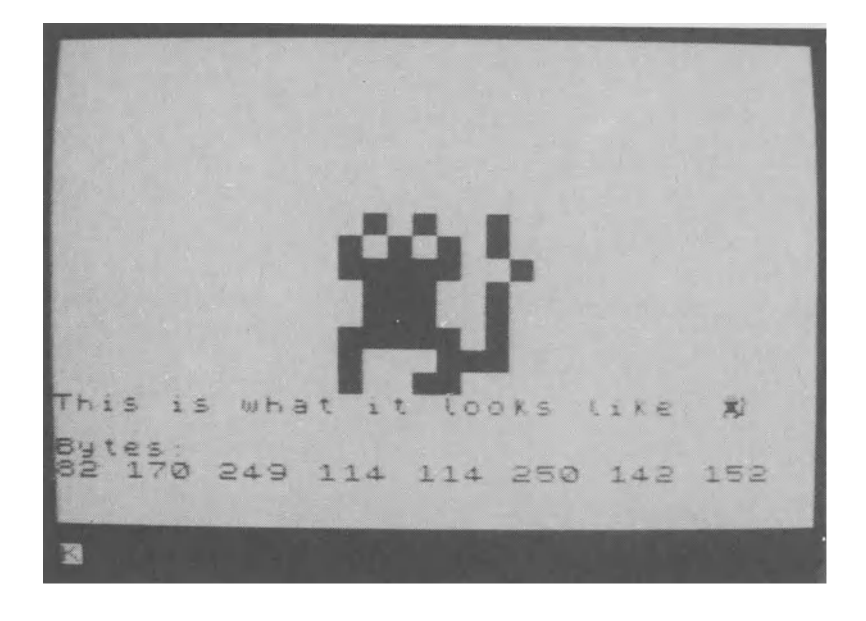
1. Die Bäume könnten Bäumen ähnlicher sehen. Experimentieren Sie mit ZEICHENKONSTRUKTEUR, um eine Fichte oder eine Tanne hervorzubringen. Und wenn Sie schon dabei sind: Wie wäre es damit, dem Skifahrer zwei Spuren zu geben und auch gegen den schrecklichen Skilift etwas zu unternehmen?
2. Die Zahl der Bäume ist mit 30 festgesetzt. Geben Sie dem User die Möglichkeit zu einem leichteren oder schwierigeren Spiel.
3. In Zeile 135 in der Schleife steht "Pause 2", um eine vernünftige Abfahrtsgeschwindigkeit zu ermöglichen. Sie könnten dem Benutzer die Möglichkeit geben, das zu ändern. (Ich meine nicht direkt: Stellen Sie eine Reihe möglicher PAUSE-Werte auf und verschaffen Sie dem Benutzer die Möglichkeit, eine Schwierigkeitsstufe von meinetwegen 1–5 einzugeben, wobei jede Stufe einen anderen PAUSE-Wert auswählt.)
4. Warum die Schwierigkeit nicht automatisch dadurch abändern, daß Sie das ganze Programm in eine Schleife stellen und die Zahl der Bäume und/oder der Abfahrtsgeschwindigkeit jedesmal ändern, wenn der Benutzer eine erfolgreiche Abfahrt hinter sich hat?

ZEICHENKONSTRUKTEUR

Benutzergewählte Grafik, leicht gemacht.

Mit diesem Programm können Sie in einem großen Maßstab auf dem Fernsehbildschirm ein Grafikzeichen entwerfen und es dann als benutzergewähltes Zeichen entsprechend einem Buchstaben Ihrer Wahl eingeben. Außerdem zeigt es den Inhalt der acht Bytes dadurch an, daß es die Reihen des Zeichens für eine Wiederverwendung bei künftigen Programmen angibt.

```
10  DIM k (8, 8)
20  FOR i = 8 TO 15
30  PRINT AT i, 12; "........"
40  NEXT i
50  LET x = 0: LET y = 0
55  GO SUB 500
60  INPUT "Pixelwert □"; v
65  IF v < > 0 AND v < > 1 THEN GO TO 60
70  LET k (x + 1, y + 1) = v
```



```
 8Ø  PRINT AT x + 8, y + 12; INVERSE v; "□"
 9Ø  LET y = y + 1
1ØØ  IF y = 8 THEN LET y = Ø: LET x = x + 1
11Ø  IF x = 8 THEN LET x = Ø: LET y = Ø: GO TO 2ØØ
12Ø  GO TO 55
2ØØ  INPUT "Ist das richtig?"; q$
21Ø  IF q$ = "" THEN GO TO 2ØØ
22Ø  IF q$ (1) = "y" THEN GO TO 6ØØ
23Ø  INPUT "Cursor mit Pfeiltasten bewegen"; j$
235  GO SUB 5ØØ
24Ø  IF INKEY$ < > "" THEN GO TO 24Ø
25Ø  IF INKEY$ = "" THEN GO TO 25Ø
26Ø  LET i$ = INKEY$
27Ø  IF i$ = "Ø" OR i$ = "1" THEN LET k (x + 1, y + 1) = VAL i$:
     PRINT AT x + 8, y + 12; INVERSE VAL i$; "□" GO TO 2ØØ
275  PRINT AT x + 8, y + 12; INVERSE k (x + 1, y + 1); "□"
28Ø  IF i$ = "5" THEN LET y = y - 1 + (y = Ø)
29Ø  IF i$ = "6" THEN LET x = x + 1 - (x = 7)
3ØØ  IF i$ = "7" THEN LET x = x - 1 + (x = Ø)
31Ø  IF i$ = "8" THEN LET y = y + 1 - (y = 7)
32Ø  GO SUB 5ØØ
33Ø  GO TO 24Ø
5ØØ  PRINT AT x + 8, y + 12; FLASH 1; INK 2; "*": RETURN
6ØØ  INPUT "Welcher Buchstabe?"; f$
61Ø  FOR n = 1 TO 8
62Ø  LET t = k (n, 1)
63Ø  FOR j = 2 TO 8
64Ø  LET t = 2 * t + k (n, j)
65Ø  NEXT j
66Ø  POKE USR f$ + n - 1, t
67Ø  NEXT n
```

```
7ØØ  PRINT "So sieht das aus; □"; CHR$ (CODE f$ + 47)
71Ø  PRINT,, "Bytes"
72Ø  FOR n = 1 TO 8
73Ø  PRINT PEEK (USR f$ + n − 1); "□"
74Ø  NEXT n
```

Programmhinweise

1 Zunächst zeigt der Bildschirm ein 8×8-Feld aus Punkten und einen blinkenden Cursor; verlangt wird ein Pixelwert. Sie geben Ø oder 1 ein: Bei Ø wird ein Punkt gelöscht, bei 1 wird er als Quadrat eingeschwärzt. Der Cursor läuft automatisch, Reihe für Reihe, durch das ganze Feld; Ihre Eingaben bauen das Zeichen nach Ihrem ersten Versuch auf.

2 Dann wird gefragt: "Ist das richtig?" Wenn Sie "j" eingeben oder irgend etwas, das mit "j" beginnt, fährt der Computer fort, das Zeichen zu laden (siehe Anmerkung 3 unten). Wenn Sie etwas anderes eingeben, erscheint die Meldung "Cursor mit Pfeiltasten bewegen". Sie müssen jetzt ENTER drücken, dann taucht der Cursor wieder auf. Er kann über das Keyboard mit den Tasten 5, 6, 7, 8 in der jeweiligen Pfeilrichtung gesteuert werden. Sobald Sie ihn an die richtige Stelle geführt haben, löscht eine Eingabe Ø oder 1 vom Keyboard aus das entsprechende Quadrat oder schwärzt es ein. Die Meldung "Ist es so richtig?" taucht wieder auf, und Sie können, wenn Sie wollen, neue Veränderungen vornehmen.

3 Sobald es wirklich richtig *ist* und Sie "j" getippt haben, fragt der Computer Sie, welchem Buchstaben Ihr Zeichen entsprechen soll. Denken Sie daran, daß jedes vom Benutzer gewählte Grafikzeichen durch die Verwendung eines Buchstabens von der Tastatur aus zugänglich ist (ausgenommen v, w, x, y, z). Welchen Sie haben wollen, müssen Sie entscheiden. Beispielsweise könnten Sie "s" nehmen: Ihr neues Zeichen ist dann an der richtigen Stelle gespeichert, so daß s im Grafik-Modus es hervorbringt. Sie können es auch mit dem Code aufrufen als CHR$ (162).

4 Der Computer führt auch die acht Bytes auf, die den Reihen des Zeichens entsprechen. Wenn Sie sich die Zahlen notieren, können sie später mühelos dadurch wieder gesetzt werden, daß man sie, wie auf Seite 127 beschrieben, mit POKE an ihren Platz stellt.

5 Von jetzt an bleibt Ihr neues Zeichen als "s" im Grafikmodus gespeichert, bis Sie den Computer abschalten. Sie können andere Zeichen an anderen Stellen laden, wenn Sie das Programm mit RUN erneut fahren.

6 Damit Sie prüfen können, ob alles richtig ist, erhalten Sie eine Probeanzeige Ihres neuen Zeichens. Wenn es Ihnen nicht gefällt, geben Sie GO TO 2ØØ. Das Zeichen wird nicht angezeigt, aber wenn Sie mit dem Cursor durch den Bereich fahren, tauchen die Quadrate wieder auf, und Sie können erneut abändern.

SCHEIBENSCHIESSEN

An einem windigen Tag auf dem Schießplatz: Wieviele Ringe schaffen Sie?

```
 10  LET d = 100
 20  LET w = 40 * (.5 − RND)
 30  LET v = 1000
 40  LET s = 0
 50  LET c = 0
100  FOR i = 1 TO 8
110  CIRCLE 127, 95, 8 * i
120  NEXT i
200  IF c = 11 THEN STOP
205  PRINT AT 20, 0; "RICHTHOEHE =";
210  INPUT e: PRINT e
220  PRINT "Abweichung =";
230  INPUT f: PRINT f
240  LET e = e * PI / 180
250  LET f = f * PI / 180
300  LET t = d / (v * COS e)
310  LET h = v * t * SIN e − 4.9 * t * t
320  LET k = w * d / 100 + d * SIN f
330  LET h0 = 8 * h + 95 + 40 * (.5 − RND)
340  LET k0 = 8 * k + 127 + 40 * (5 − RND)
350  IF h0 < 0 OR h0 > 175 OR k0 < 0 OR k0 > 255
     THEN GO TO 500
360  GO SUB 390
370  PRINT AT 20, 0;
     "□□□□□□□□□□□□□□□□",,
     "□□□□□□□□□□□□□□□□"
     [2 × 16 Leerst.]
```

```
375  LET c = c + 1
38Ø  GO TO 2ØØ
39Ø  INK 3
4ØØ  PLOT k Ø – 8, h Ø: DRAW 16, Ø: PLOT k Ø, h Ø – 8: DRAW Ø, 16
4Ø5  BEEP .1, 5
41Ø  CIRCLE k Ø, h Ø, 6
415  IF c < 1 THEN GO TO 48Ø
42Ø  LET q = SQR ((k Ø – 127) * (k Ø – 127) + (h Ø – 95) * (h Ø – 95))
43Ø  LET q = INT (q / 8)
44Ø  LET q = 1ØØ – 1Ø * q
45Ø  IF q < 3Ø THEN LET q = Ø
46Ø  LET s = s + q
47Ø  PRINT AT Ø, 25; c; TAB 28; s
48Ø  INK Ø
49Ø  RETURN
5ØØ  PRINT AT Ø, Ø; "Außerhalb Bildschirm"
51Ø  PAUSE 5Ø
52Ø  PRINT AT Ø, Ø; "□□□□□□□□□□" [1Ø Leerst.]
53Ø  PRINT AT 2Ø, Ø;
     "□□□□□□□□□□□□□□□□"
     "□□□□□□□□□□□□□□□□"
     [2 × 16 Leerst.]
535  PRINT AT Ø, 25; c
54Ø  LET c = c + 1
55Ø  GO TO 2ØØ
```

Programmhinweise

1 Nach RUN werden Sie aufgefordert, eine Richthöhe (in Grad) einzugeben. Irgendwo nahe bei Null ist am besten, etwa zwischen − 2 und + 2.

2 Als nächstes sollen Sie eine seitliche *Abweichung* (als Ausgleich für Seitenwind) eingeben. Der Wert sollte zwischen − 1Ø und + 1Ø liegen. Achtung: Sie bekommen nicht mitgeteilt, von welcher Seite der Wind weht!

3 Sie haben einen Schuß zum Einschießen, der nicht zählt, dann 1Ø Schüsse. Der Wind wechselt zwischen den Schüssen leicht.

4 Das Ergebnis und die Zahl der verbrauchten Schüsse werden oben rechts angezeigt – zuerst die Zahl der Schüsse, dann die Zahl der Ringe.

Fehlermeldungen des ZX Spectrum

0	a) Alles ok b) Sprung zu einer Zeilennummer, die größer ist als jede existierende
1	**NEXT** ohne **FOR**
2	Variable nicht gefunden
3	Index der Variablen oder des Substrings falsch
4	Speicherplatz reicht nicht aus für momentanes Vorhaben
5	Anzeige außerhalb des Bildschirms
6	Numerischer Überlauf (Zahl größer als 10^{38})
7	**RETURN** ohne **GO SUB**
8	Ende des Files (Microdrive)
9	**STOP** ausgeführt
A	Ungültiges Argument
B	Ganzzahl außerhalb des zulässigen Bereichs
C	Argument von **VAL** oder **VAL$** ergibt ungültigen Ausdruck
D	Abbruch bei Bandoperationen oder beim Drucken durch **BREAK**
E	**DATA**-Liste zu Ende
F	Ungültiger File-Name
G	Kein Platz mehr für neue Programmzeile
H	**STOP** in einem **INPUT**-Befehl
I	**FOR** ohne **NEXT**
J	Fehler bei I/O-Daten (Microdrive)
K	Ungültige Farbe
L	Programmabbruch mit **BREAK**
M	**RAMTOP** zu groß oder zu klein
N	Sprung zu einem nicht mehr existierenden Befehl
O	Ungültiger Strom (Microdrive)
P	**FN** ohne Definition
Q	Falsches Argument in **FN**
R	File auf Band gefunden, kann aber nicht geladen werden